**FLA大作业实验报告**

221220092谷莘

1. PDA部分
2. 解析器

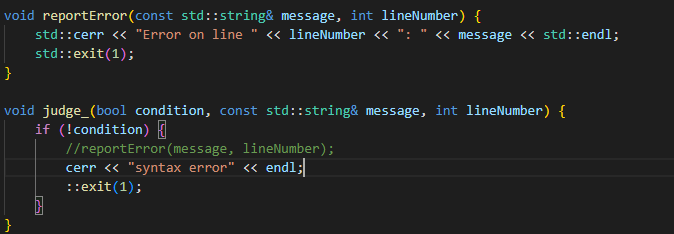
解析器loadPDAFromFile实现的逻辑如下：

从指定文件中读取PDA配置，通过文件流读取每一行数据，跳过空行和注释行，并且使用正则表达式和字符串操作解析状态集(Q)、输入符号集(Σ)、栈符号集(Γ)、初始状态(q0)、栈起始符号(z0)和终结状态集(F)（分别对应states, inputSymbols, stackSymbols, initialState, stackStartSymbol, finalStates, transitions）；

检查每个部分的格式正确性，确保它们被正确解析和存储在PDA类的相应属性中。

将转移函数(δ)存储到PDA对象的transitions向量中。

对不正确的语法格式，我们会给出syntax error的报错（支持打印错误位置的模式）：



1. 模拟器

模拟器用解析器解析出的PDA类进行模拟，PDA类的方法simulate的逻辑如下：

使用simulate方法接受输入字符串，并根据PDA的定义进行处理；

初始化栈，压入栈起始符号；

逐字符读取输入字符串，检查每个字符是否在输入符号集中；

根据当前状态、输入符号和栈顶符号遍历转移函数，寻找匹配的转移；

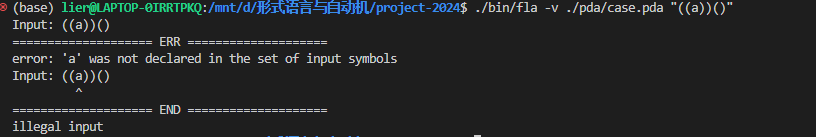
更新当前状态和栈内容，对ε转移（用"\_"表示），对遍历的index不增加；

模拟结束后，检查当前状态是否为终结状态，并确认栈是否为空，以判断输入字符串是否被接受；

1. Verbose模式

实现了加分点PDA模拟的verbose模式，打印的内容包括当前输入字符、状态和栈内容；在输入非法时，能定位非法输入的位置，效果演示如下：





1. TM部分
2. 解析器

解析器在TM中的readFile方法中实现，大致逻辑如下：

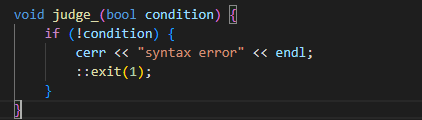
从指定文件中读取TM配置；

解析状态集、输入符号集、磁带符号集、初始状态、终结状态集、磁带数量和转移规则（分别对应states, stateLookupTable, inputSymbols, tapeSymbols, currentState, finalStates, numberOfTapes, tapeCollection, transitionRules）；

使用字符串流和分割函数解析各部分内容（使用实例splitString）；

检查每个部分的格式正确性，确保它们被正确解析和存储在TM类的相应属性中；

对不正确的语法格式，我们会把判断扔给judge\_函数，如果对应的语法条件不满足，我们会在cerr中抛出syntax error:



1. 模拟器

我们除了图灵机所代表的类TM, 还创建了纸带类Tape、transition对应的类Rule作为子类，模拟时，通过调用TM的run类进行模拟，run中对输入错误进行检查，随后调用executeTuringMachine类启动模拟，其流程如下：

初始化磁带（Tape），并设置初始状态；

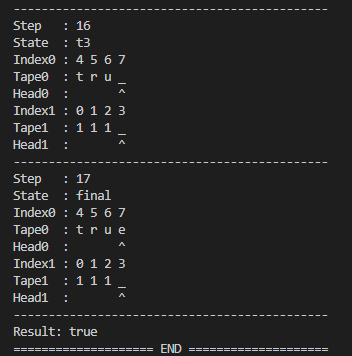
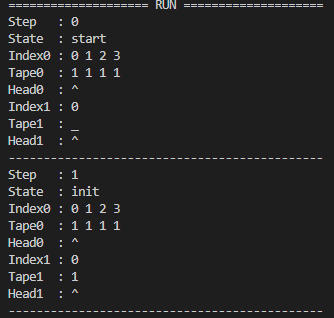
在每个步骤中，根据当前状态和磁带符号选择合适的transition，并应用；

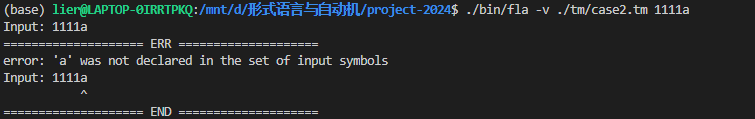
更新当前状态、磁带内容和磁头位置；

检查是否达到终结状态，以判断输入字符串是否被接受;

1. Verbose 模式

我们完成了verbose模式，并且完成了对齐要求，效果如下：





1. 样例编写
2. case.pda

这一PDA的目标是匹配任意可能的括号组合。我的思路是将所有左括号直接入栈，碰到右括号消除；如果右括号没有与之匹配的左括号，拒绝；如果正好栈空，我们接受。

#Q = {q,accept,reject}

#S = {(,)}

#G = {(,),Z}

#q0 = q

#z0 = Z

#F = {accept}

q ( Z q (Z

q ( ( q ((

q ) ( q \_

q ) Z reject Z

q \_ Z accept \_

1. case1.tm

这个TM的作用是对给出的a^ib^j,打印出c^(i\*j);我这里给出我的大致思路，具体过程见代码：

我一共用了三条纸带，第一条用来存结果，第二条用来放置a,第三条放置c。我们开始时读取到a,我们将它搬到纸带2上；当读到b时，我们将与纸带2上a相同数量的c放在纸带3的后端；在读取b时如果有碰到a,我们打印illegal\_input；在都读取完时，我们将纸带3上的c搬到纸带1上即可；

1. case2.tm

这个TM的作用是数平方数个数的1，我的思路如下：

我的思路本质是在原来的1计数上依次减去1,3,5,7......每次减的时候，如果正好够减那么是平方数；如果1还剩了，那么被减数加2继续减；如果1不够减，那么不是平方数。

这里放出一张设计简图：

