# 实验内容

完成 PythonT 语法分析器的设计，包括Bison(YACC)实现与C++实现。

# Bison(YACC)实现

## l文件

### 初始化

图 2‑1 初始化代码

初始化部分，引入标准库与计数器，创建缩进栈。定义特殊全局变量last\_token\_was\_newline指示是否需要移进NEWLINE。

### 规则预定义

图 2‑2 规则预定义代码

为了避免影响代码可读性，按照PythonT各类单词组成进行规则预定义,此外设置缩进处理模式INDENT\_PROCESSING。

需要注意的是，实际测试案例中有一处与实验平台所述规则存在出入：

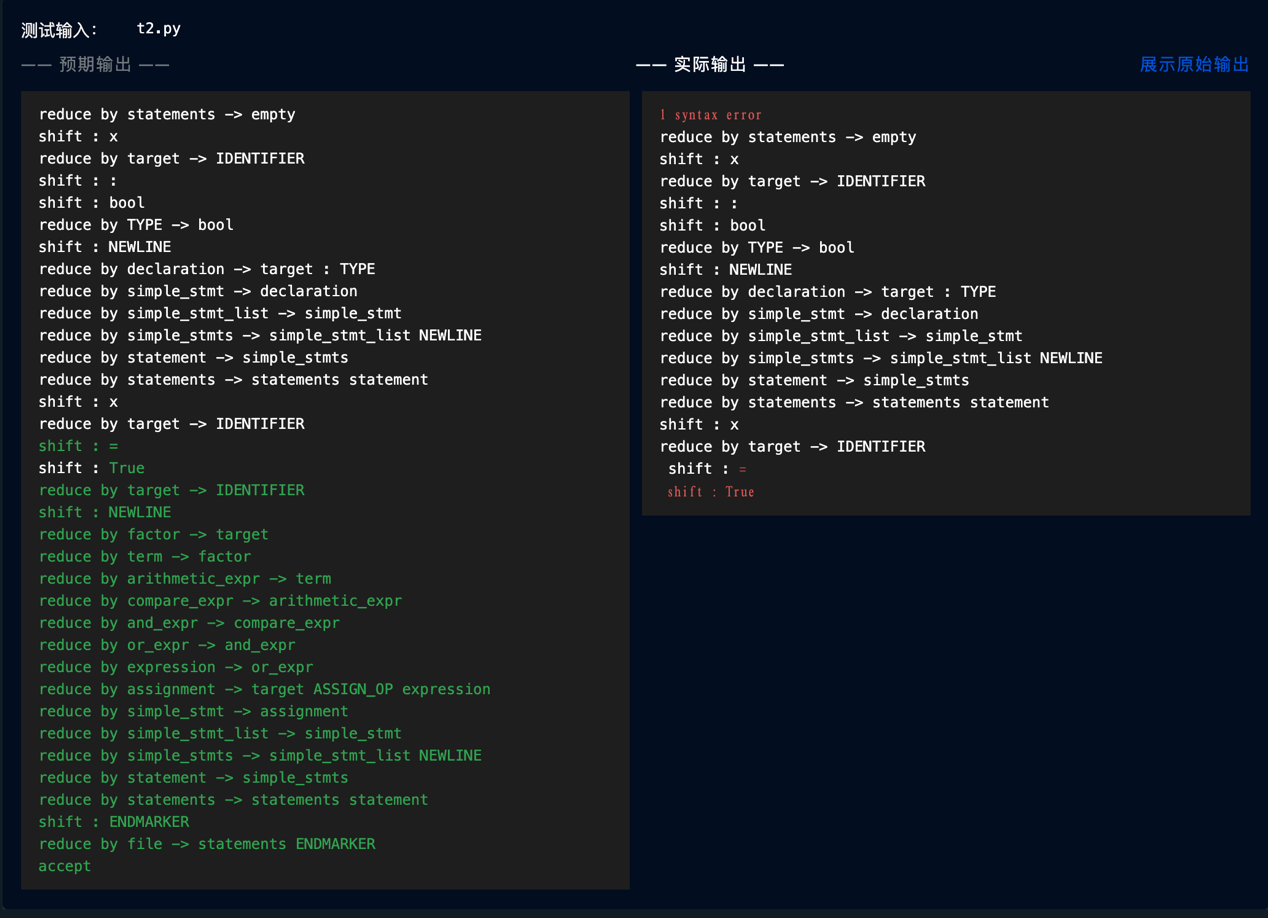
* True和False在规则中被归类为关键字，实际测试案例中按照标识符处理；

图 2‑3 将True归类为标识符示例

### 规则执行

图 2‑4 规则执行代码

左侧为规则对应的正则表达式形式定义，右侧为正则表达式匹配到后执行的操作，条目声明的先后代表处理优先级。

图 2‑5 缩进处理操作

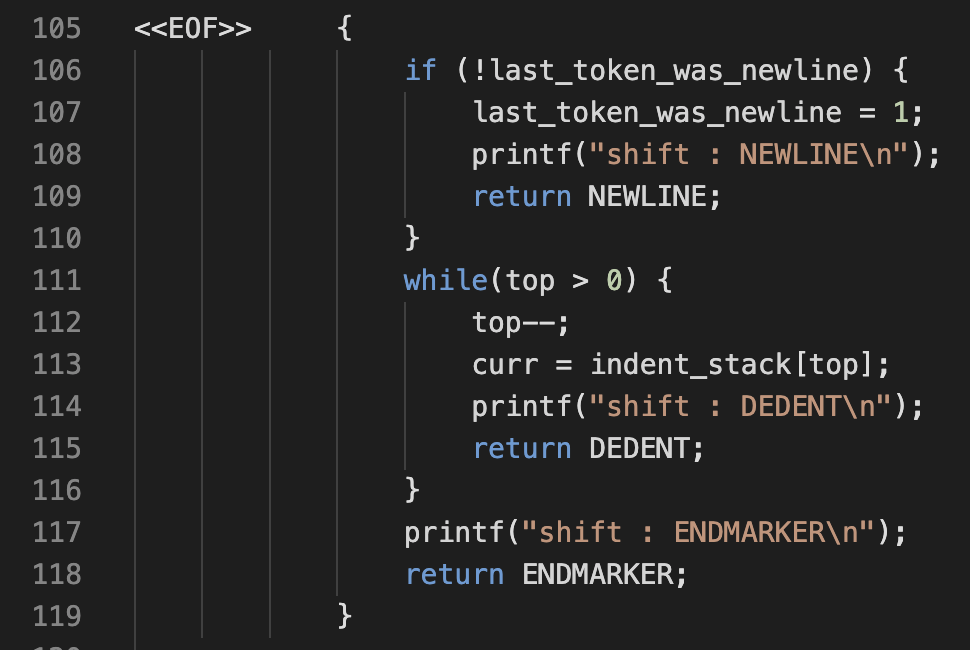
缩进处理逻辑中，对空格和缩进进行匹配，通过判断缩进长度与缩进栈顶大小关系决定是入栈（输出INDENT记录）还是出栈（输出DEDENT记录）。如果出栈时栈中没有与当前缩进长度相同元素，则输出indentation error。

图 2‑6 文件末尾处理

文件末尾输出NEWLINE，还原缩进并输出ENDMARKER。

## y文件

### 初始声明

图 2‑7 初始声明代码

**头文件声明**

包含了常见的 C 头文件；

yylineno：记录当前行号（通常由 Lex 提供）；

yyin：词法分析器输入流；

last\_token\_was\_newline：用于处理缩进的判断逻辑；

yyerror()：语法出错的处理函数；

yylex()：词法分析器函数；

**%union和token声明**

union 表示语法分析过程中所有符号可能拥有的数据类型；

定义终结符（token），有的携带语义值（如 IDENTIFIER 是 char\*）；

ADD\_OP, MUL\_OP, REL\_OP, ASSIGN\_OP 是代表 +, \*, ==, = 等操作的泛型 token。

**优先级和结合性声明**

为表达式优先级和结合性设定规则：

\* 高于 +

==（REL\_OP）不能连续出现（%nonassoc 表示不能连用）

### 语法规则部分

略，参照题目描述进行编写即可。

# C++实现

实现print语句（简化版）的语法分析，，要求程序支持简单的语法错误恢复功能。

## 初始化与主函数

图 3‑1 初始化与主函数代码

引入完成操作所必要的库，定义Token结构体，包括类型与值两个属性。主函数部分，读取缓冲区后的输入首先交给词法分析器函数得到token序列，然后使用语法分析器类对token进行LR(1)分析。

## 词法分析器

图 3‑2 词法分析器函数

函数仅进行print语句所必需的词法分析，返回动态长度的token序列。

## 语法分析器

### 文法

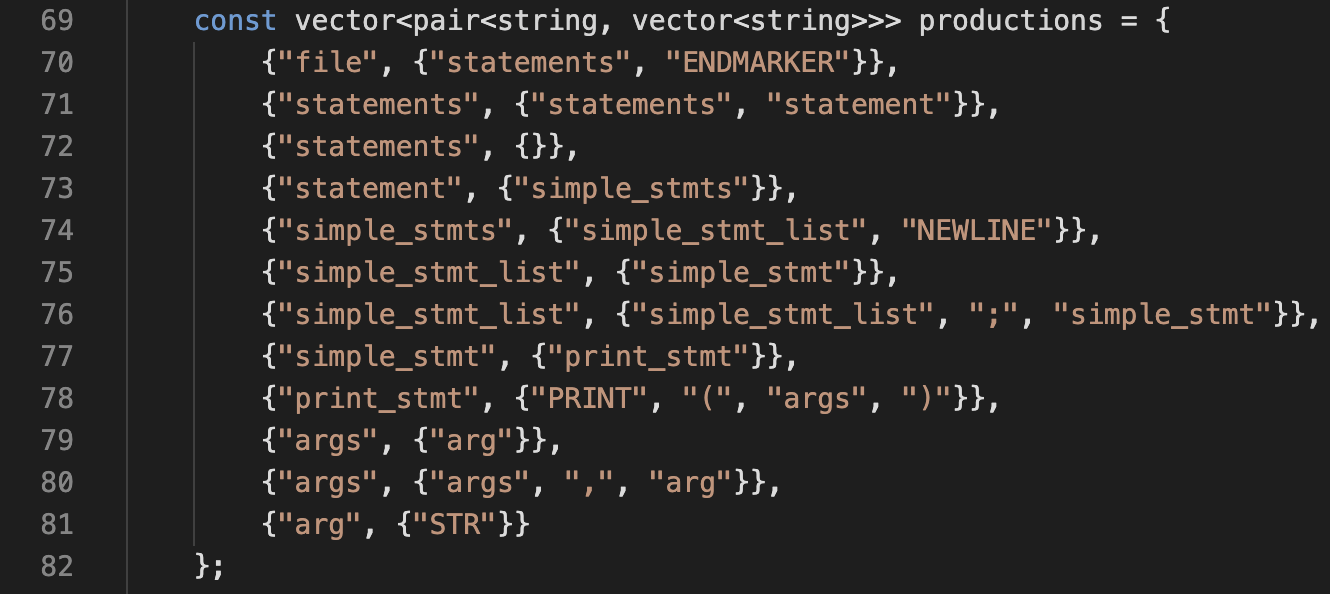


图 3‑3 文法定义

### 状态转移

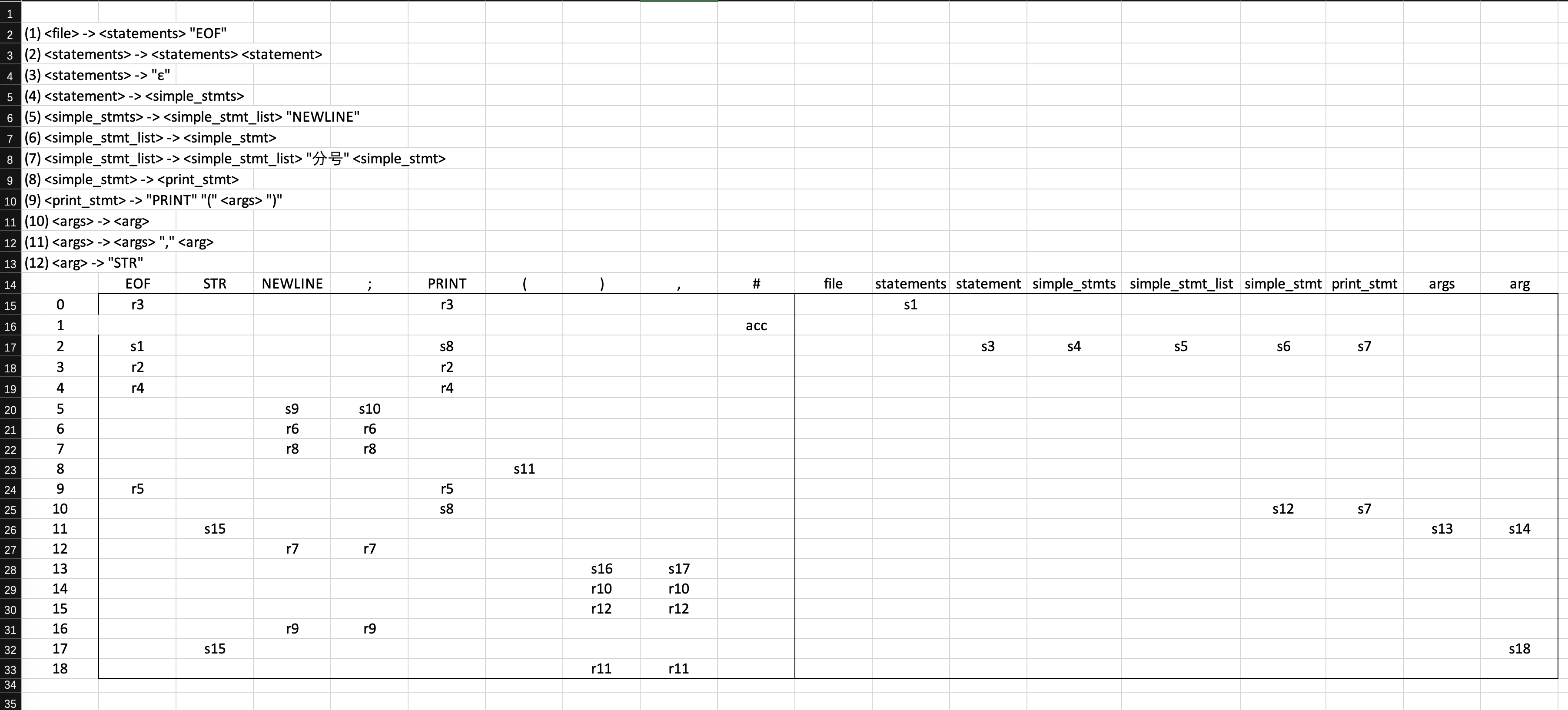


图 3‑4 状态转移表

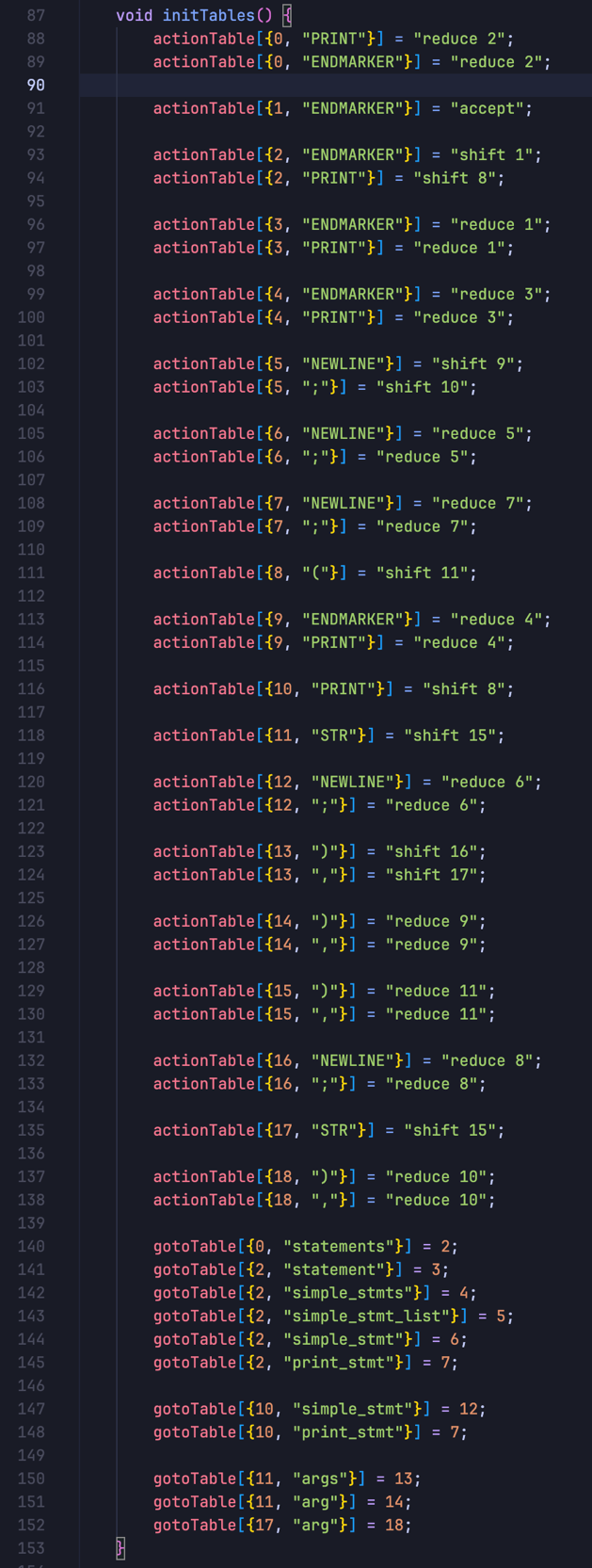
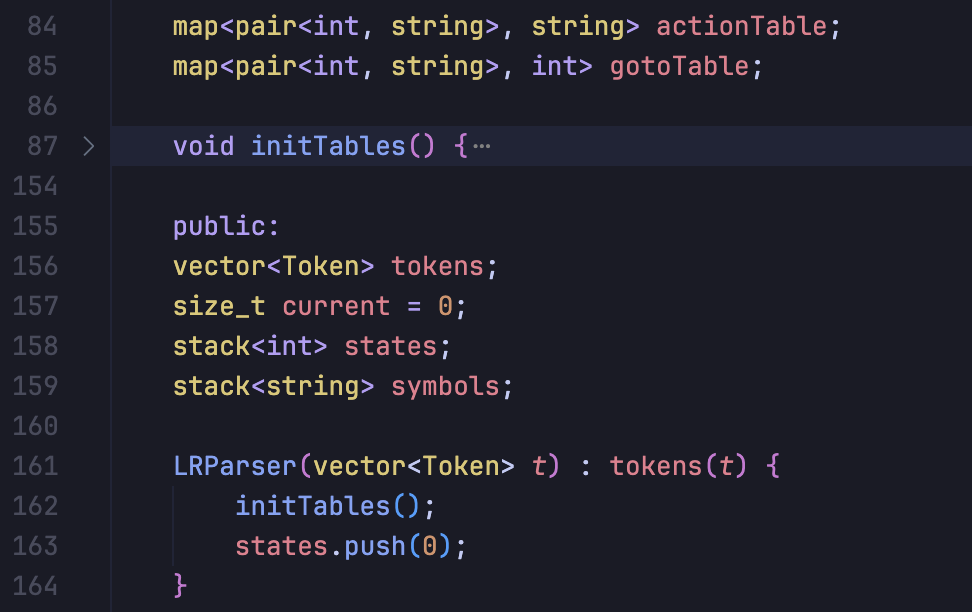
由文法可得状态转移表，依据转移表进行C++程序当中定义编写：

图 3‑5 状态转移表定义

### 语法分析



该方法使用：

* actionTable：存储 shift, reduce, accept 动作
* gotoTable：规约后用于转移状态
* states 栈：存储状态编号
* symbols 栈：存储文法符号





**LR分析逻辑：**

1. 当前状态 + 当前 token => 查表得到操作（shift/reduce）
2. 执行对应操作：
   1. shift：推进 token，状态栈推进
   2. reduce：根据产生式回退栈，更新符号栈并GOTO新状态
   3. accept：成功解析

**错误处理机制：**

* 若查表失败，输出错误信息。
* 启用“恐慌模式”跳过错误 token，直到找到可以恢复的状态。

# 线上测试用例通过情况

Yacc第20组测试用例始终没有找到通过的方法。



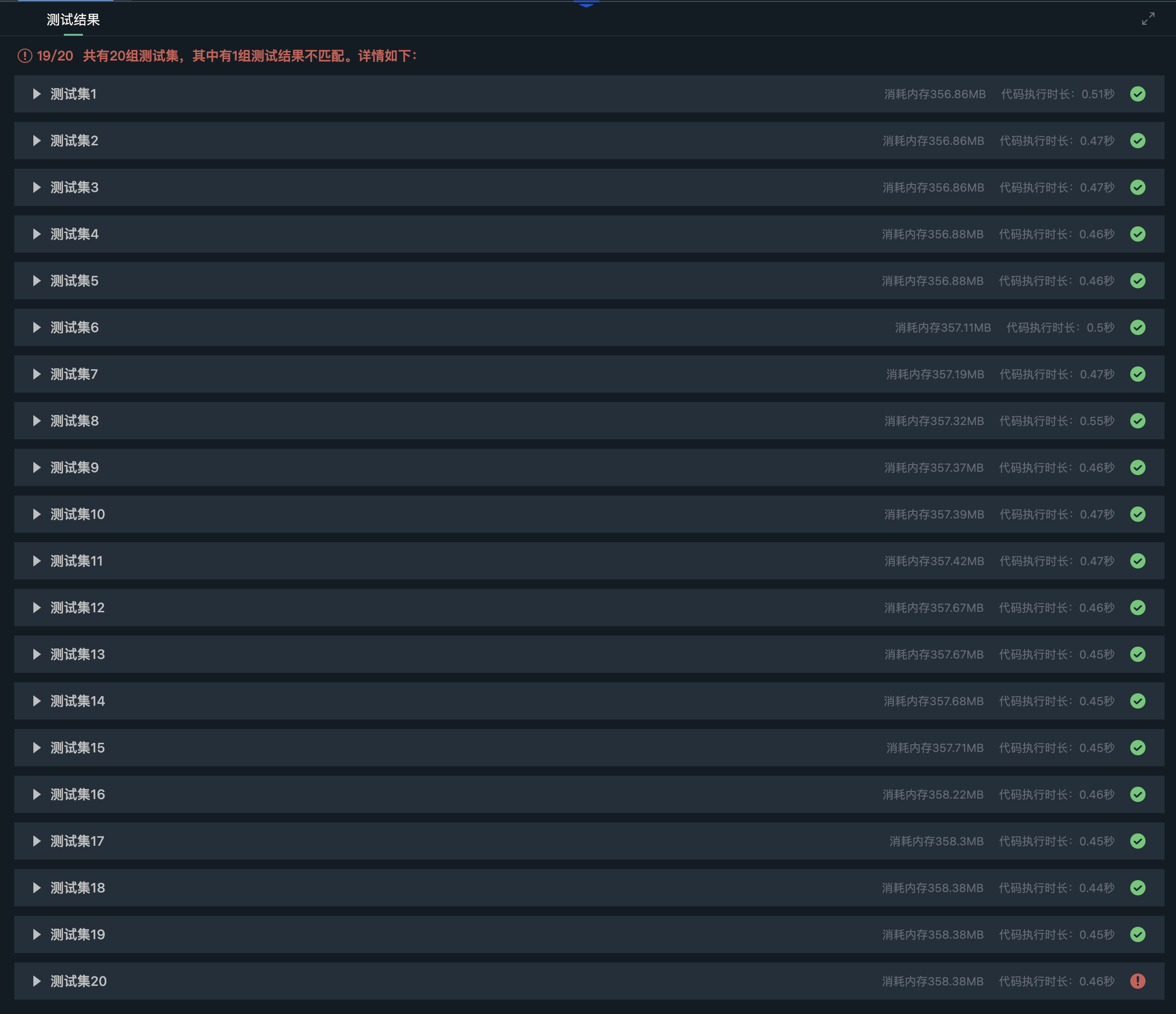


图 4‑1 yacc程序通过情况



图 4‑2 C++程序通过情况