|  |  |
| --- | --- |
| **成绩** |  |

**2025年《程序设计语言原理》**

**实验报告**

**班级： 34082301**

**姓名： 高浚然**

**学号： 2023215319**

**指导老师： 陈霖**

**实验名称： Lab4**

**实验地点： A503**

## 实验四：

1. **实验题目**

解析LR(0) Parser Table

1. **实验目的**

**使用Java语言实现给定CFG（规约步骤）和LR(0) Parser Table，检验给定输入字符串是否可被接受的流程。**

1. **实验内容**

**1. 构造解析表的Java表示；**

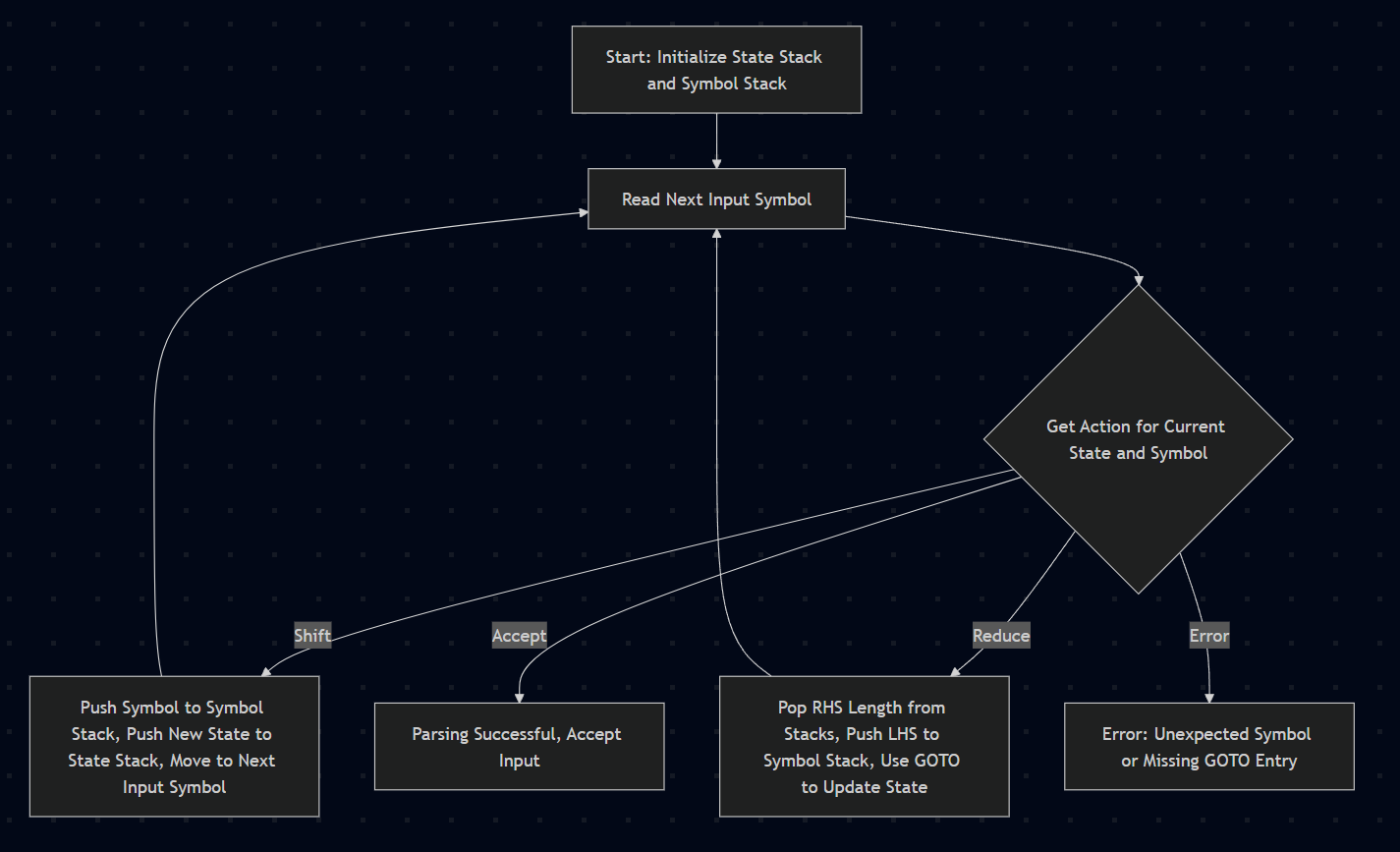
**2. 构造输入；**

**3. 构造函数shift 和 reduce；**

**4. 循环解析。**

1. **实验步骤**

代码的核心逻辑：



1. **初始化**
   * **状态栈 (State Stack)**：将初始状态 s0 压入状态栈。
   * **符号栈 (Symbol Stack)**：初始化为空，用于存储正在处理的语法符号。
2. **读取输入符号**

逐个读取输入串中的符号，通常以 $ 结尾表示输入结束。

1. **根据 ACTION 表判断下一步操作**

**移入 (shift)**：如果 ACTION 表中对应的状态和符号是 sN，则：将当前符号压入符号栈；将 N 作为新的状态压入状态栈；移动到下一个输入符号。

**规约 (reduce)**：如果 ACTION 表返回 rN（对应某条产生式），则：弹出符号栈和状态栈中的对应符号长度。将产生式的左部（非终结符）压入符号栈。根据 GOTO 表查找下一状态，并压入状态栈。

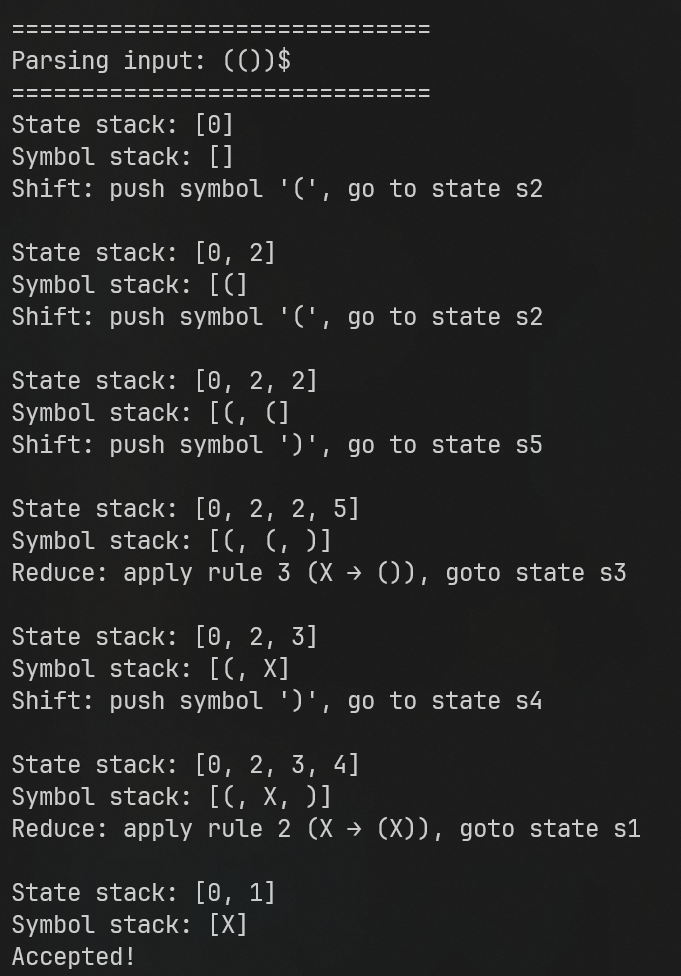
**接受 (accept)**：如果 ACTION 表返回 acc，表示分析成功，接受输入。

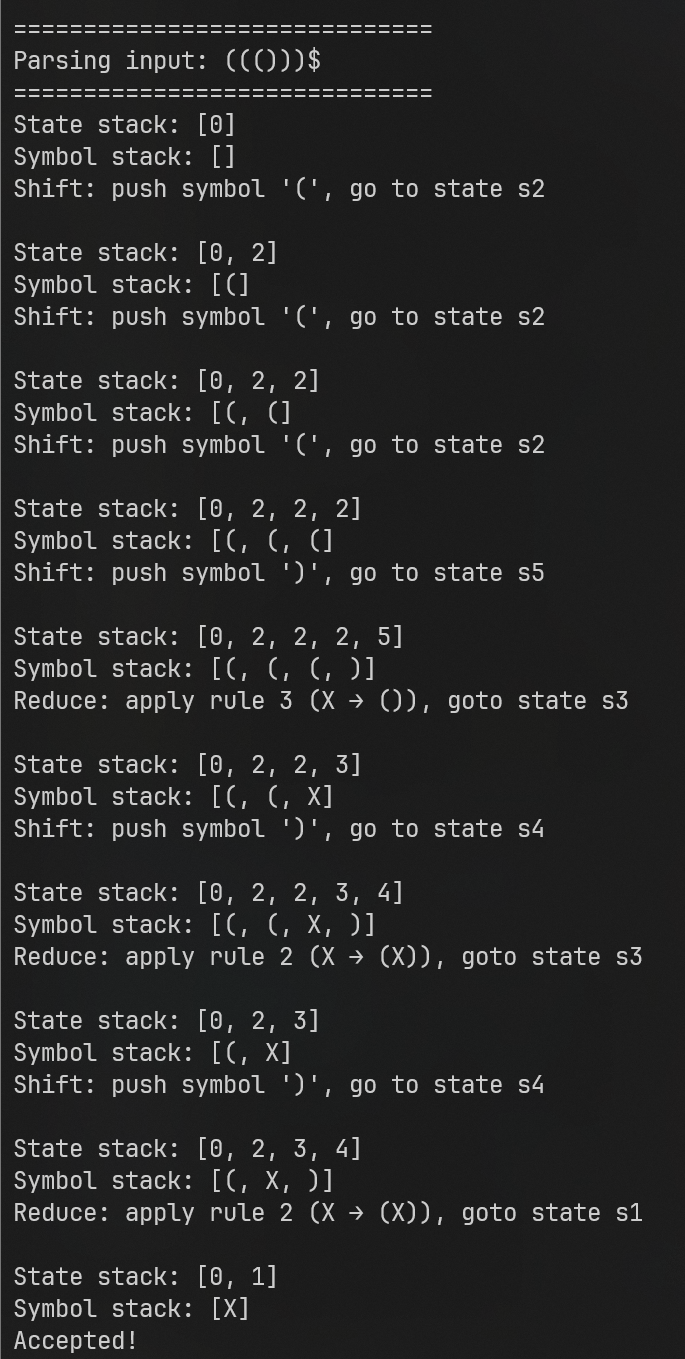
**错误处理**：如果 ACTION 表返回空或找不到对应的 GOTO 项，报告语法错误。

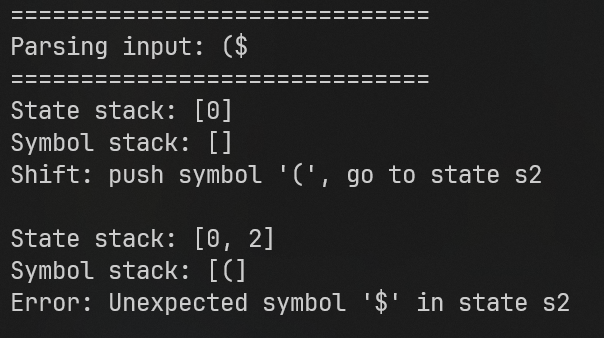
1. **重复步骤 2-3，直到输入被完全接受或发生语法错误**。
2. **实验结果与分析**

以下是代码对于三个给定字符串的输出：

每步中，我给出了当前栈的状态和当前操作。







输出结果显示，前两个输入可被接受，第三个输入不可被接受。

1. **核心代码清单（只写代码核心）**

import java.util.\*;

public class BottomUpParser {

static class Rule {

String lhs;

String rhs;

Rule(String lhs, String rhs) {

this.lhs = lhs;

this.rhs = rhs;

}

}

static final Rule[] rules = {

new Rule("S", "X"),

new Rule("X", "(X)"),

new Rule("X", "()")

};

static final Map<Integer, Map<String, String>> actionTable = new HashMap<>();

static {

actionTable.put(0, Map.of("(", "s2"));

actionTable.put(1, Map.of("$", "acc"));

actionTable.put(2, Map.of("(", "s2", ")", "s5"));

actionTable.put(3, Map.of(")", "s4"));

actionTable.put(4, Map.of("(", "r2", ")", "r2", "$", "r2"));

actionTable.put(5, Map.of("(", "r3", ")", "r3", "$", "r3"));

}

static final Map<Integer, Map<String, Integer>> gotoTable = new HashMap<>();

static {

gotoTable.put(0, Map.of("X", 1));

gotoTable.put(2, Map.of("X", 3));

}

static Stack<Integer> stateStack = new Stack<>();

static Stack<String> symbolStack = new Stack<>();

public static void main(String[] args) {

String[] testInputs = { "(())$", "((()))$", "($" };

for (String input : testInputs) {

System.out.println("\n==============================");

System.out.println("Parsing input: " + input);

System.out.println("==============================");

parseInput(input);

}

}

static void parseInput(String input) {

stateStack.clear();

symbolStack.clear();

stateStack.push(0);

int index = 0;

while (true) {

int state = stateStack.peek();

String symbol = String.valueOf(input.charAt(index));

String action = getAction(state, symbol);

printStacks();

if (action == null) {

System.out.println("Error: Unexpected symbol '" + symbol + "' in state s" + state);

return;

}

if (action.equals("acc")) {

System.out.println("Accepted!");

return;

}

if (action.startsWith("s")) {

int newState = Integer.parseInt(action.substring(1));

shift(newState, symbol);

index++;

} else if (action.startsWith("r")) {

int ruleIndex = Integer.parseInt(action.substring(1)) - 1;

reduce(ruleIndex);

} else {

System.out.println("Error: Invalid action '" + action + "'");

return;

}

System.out.println(); // 空行

}

}

static void shift(int newState, String symbol) {

symbolStack.push(symbol);

stateStack.push(newState);

System.out.println("Shift: push symbol '" + symbol + "', go to state s" + newState);

}

static void reduce(int ruleIndex) {

Rule rule = rules[ruleIndex];

int popCount = rule.rhs.length();

for (int i = 0; i < popCount; i++) {

if (!symbolStack.isEmpty()) symbolStack.pop();

if (!stateStack.isEmpty()) stateStack.pop();

}

symbolStack.push(rule.lhs);

int currentState = stateStack.peek();

Map<String, Integer> gotoRow = gotoTable.get(currentState);

if (gotoRow == null || !gotoRow.containsKey(rule.lhs)) {

System.out.println("Error: No GOTO entry for state s" + currentState + " and symbol '" + rule.lhs + "'");

return;

}

int newState = gotoRow.get(rule.lhs);

stateStack.push(newState);

System.out.println("Reduce: apply rule " + (ruleIndex + 1) + " (" + rule.lhs + " → " + rule.rhs + "), goto state s" + newState);

}

static String getAction(int state, String symbol) {

Map<String, String> row = actionTable.get(state);

return (row != null) ? row.getOrDefault(symbol, null) : null;

}

static void printStacks() {

System.out.println("State stack: " + stateStack);

System.out.println("Symbol stack: " + symbolStack);

}

}