《编译原理》实验报告

学　　院　计算机工程与科学学院

组　　号　10 组

实验题号　实验三

日　　期　2025 年 05 月 14 日

学号	姓名	主要工作	贡献因子
22122792	王潇	实验代码与报告	0.25
22121037	钱鹏程	实验代码	0.25
22121639	何勇乐	实验代码与报告	0.25
22121630	汪江豪	实验代码	0.25

《编译原理》实验报告

一、 实验题目

根据 PL/0 语言的文法规范，设计并实现一个语法分析程序，用于对词法分析程序输出的单词序列进行语法检查和结构分析。选择一种语法分析方法（如递归下降法、LL(1)分析法或 SLR(1)分析法），并将实验二中词法分析的结果作为输入，判断表达式的语法是否正确，对于合法表达式输出 “语法正确”，否则指出错误原因。

二、 实验内容设计与实现

本程序使用 LL(1)表驱动分析法对输入的表达式进行语法分析。通过计算 SELECT 集可以得到如表1所示的 LL(1)分析表。

Row Key	#	ident	lparen	minus	number	plus	rparen	slash	times
Y	empty	error	error	empty	error	empty	empty	slash, B, Y	times, B, Y
A	error	B, Y	B, Y	error	B, Y	error	error	error	error
B	error	ident	lparen, S, rparen	error	number	error	error	error	error
X	empty	error	error	minus, A, X	error	plus, A, X	empty	error	error
S	error	A, X	A, X	minus, A, X	A, X	plus, A, X	error	error	error

表 1　LL(1)分析表

`match_S` 和 `terminal_match` 是两个体现 LL(1)语法分析核心思想的关键函数。这两个函数共同支撑了整个语法分析器的运行逻辑，分别负责处理非终结符和终结符的匹配过程。

2.1. match_S

`match_S` 函数根据当前分析栈顶的非终结符和输入队列的第一个符号，在预定义的 LL(1)分析表中查找对应的产生式，并将该产生式的右部逆序压入分析栈中。这个函数完整体现了 LL(1)表驱动程序的核心机制。首先，它通过查表的方式决定下一步的动作，这正是 LL(1)分析法的本质特征之一；其次，由于使用的是栈结构，为了保证产生式展开顺序的正确性，必须将产生式的各个元素按逆序压入栈中。此外，该函数还加入了异常处理机制，以应对分析表中未定义的输入情况，从而提高了程序的健壮性。当发生错误时，函数会输出详细的错误信息，包括当前预期的非终结符、实际输入的符号以及其在输入流中的位置。

代码 1　match_S 函数

```
bool match_S(const string &S, stack< string > &analysis_stack,
  const unordered_map< string, unordered_map< string, vector<
string > > >
    &analysis_table,
  queue< string > &parsedData, int length_origin) {
    analysis_stack.pop();
    try {
        const vector< string > &values =
          analysis_table.at(S).at(parsedData.front()); if
(values.at(0) != "error") {
            for (auto it = values.rbegin(); it != values.rend();
++it) {
                analysis_stack.push(*it);
            }
        }
        else {
            cout << "Syntax ERROR!" << endl;
            cout << "Excepted " << S << " at position "
                 << length_origin - parsedData.size() << ", but
got "
                 << parsedData.front() << endl;
            return false;
        }
        return true;
    } catch (const std::out_of_range &e) {
        cout << "Unknown symbol! Syntax ERROR!" << endl;
        return false;
    }
}
```

2.2. terminal_match

`terminal_match` 函数的功能是判断当前输入队列中的第一个符号是否与分析栈顶的符号相匹配。如果两者相同，则将它们分别从队列和栈中弹出，表示该终结符已经被正确识别和处理；如果不匹配，则输出语法错误信息并返回 false。这个函数的设计简洁而高效，符合 LL(1)分析方法中逐个读取输入并逐步展开或匹配的执行流程。它的模块化结构使得主控程序可以清晰地调用它来处理所有终结符的情况，增强了代码的可维护性和可读性。

代码 2　terminal_match 函数

```
bool terminal_match(
  queue< string > &parsedData, stack< string > &analysis_stack)
{
    if (parsedData.front() == analysis_stack.top()) {
        parsedData.pop();
        analysis_stack.pop();
        return true;
    }
    else {
        cout << "Syntax ERROR!" << endl;
        return false;
    }
    return true;
}
```

三、 实验数据测试

3.1. 正例 1

输入如下。

```
-5/8/9*9+(5*(6+7)*7)-8/(5+6)
```

输出。

```
(minus, -)
(number, 5)
(slash, /)
(number, 8)
(slash, /)
(ident, a)
(times, *)
(number, 9)
(plus, +)
(lparen, ( )
(number, 5)
(times, *)
(lparen, ( )
(ident, b)
(plus, +)
(number, 7)
(rparen, ))
(times, *)
(ident, abc)
(rparen, ))
(minus, -)
(number, 8)
(slash, /)
(lparen, ( )
(ident, d)
(plus, +)
(number, 6)
(rparen, ))
```

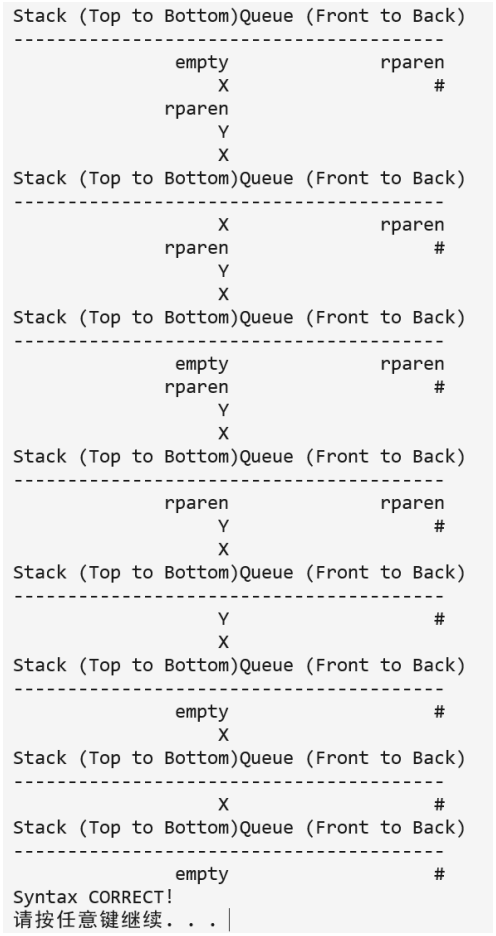

图 1　正例输出结果

3.2. 反例 1——未定义符号

输入如下，中间出现了 `++` 是未定义的符号。

```
1.  -1+2*3*(7/(5++6)-8)
```

输出。

```
(minus, -)
(number, 1)
(plus, +)
(number, 2)
(times, *)
(number, 3)
(times, *)
(lparen, ( )
(number, 7)
(slash, /)
(lparen, ( )
(number, 5)
(plus, +)
(plus, +)
(number, 6)
(rparen, ))
(minus, -)
(number, 8)
(rparen, ))
```

```
                    X                    #
Stack (Top to Bottom)Queue (Front to Back)
---------------------------------------
                    X                plus
               rparen                plus
                    Y              number
                    X              rparen
               rparen                minus
                    Y              number
                    X              rparen
                                        #
Stack (Top to Bottom)Queue (Front to Back)
---------------------------------------
                 plus                plus
                    A                plus
                    X              number
               rparen              rparen
                    Y                minus
                    X              number
               rparen              rparen
                    Y                    #
                    X
Stack (Top to Bottom)Queue (Front to Back)
---------------------------------------
                    A                plus
                    X              number
               rparen              rparen
                    Y                minus
                    X              number
               rparen              rparen
                    Y                    #
                    X
Syntax ERROR!
Excepted A at position 13, but got plus
Syntax ERROR!
请按任意键继续. . .
```

图 2　反例输出结果

3.3. 反例 2——括号不匹配

输入如下，内部出现了左右括号不匹配的现象。

```
+5/8/9*9+(5*6+7)*7)-8/(5+6)
```

输出。

```
(plus, +)
(number, 5)
(slash, /)
(number, 8)
(slash, /)
(number, 9)
(times, *)
(number, 9)
(plus, +)
(lparen, ( )
(number, 5)
(times, *)
(number, 6)
(plus, +)
(number, 7)
(rparen, ))
(times, *)
(number, 7)
(rparen, ))
(minus, -)
(number, 8)
(slash, /)
(lparen, ( )
(number, 5)
(plus, +)
(number, 6)
(rparen, ))
```

```
Stack (Top to Bottom)Queue (Front to Back)
--------------------------------------
                   X              rparen
                                   minus
                                  number
                                   slash
                                  lparen
                                  number
                                    plus
                                  number
                                  rparen
                                       #
Stack (Top to Bottom)Queue (Front to Back)
--------------------------------------
                empty              rparen
                                   minus
                                  number
                                   slash
                                  lparen
                                  number
                                    plus
                                  number
                                  rparen
                                       #
Syntax ERROR!
请按任意键继续. . .
```

图 3　反例输出结果

3.4. 反例 3——未知符号

输入如下，出现了未知符号 `@` 的情况。

```
1. -1+2*3*(7/(5+6)-8)/9+(-5)@5
```

输出。

```
(minus, -)
(number, 1)
(plus, +)
(number, 2)
(times, *)
(number, 3)
(times, *)
(lparen, ( )
(number, 7)
(slash, /)
(lparen, ( )
(number, 5)
(plus, +)
(number, 6)
(rparen, ))
(minus, -)
(number, 8)
(rparen, ))
(slash, /)
(number, 9)
(plus, +)
(lparen, ( )
(minus, -)
(number, 5)
(rparen, ))
(Unknown symbol, @)
(number, 5)
```

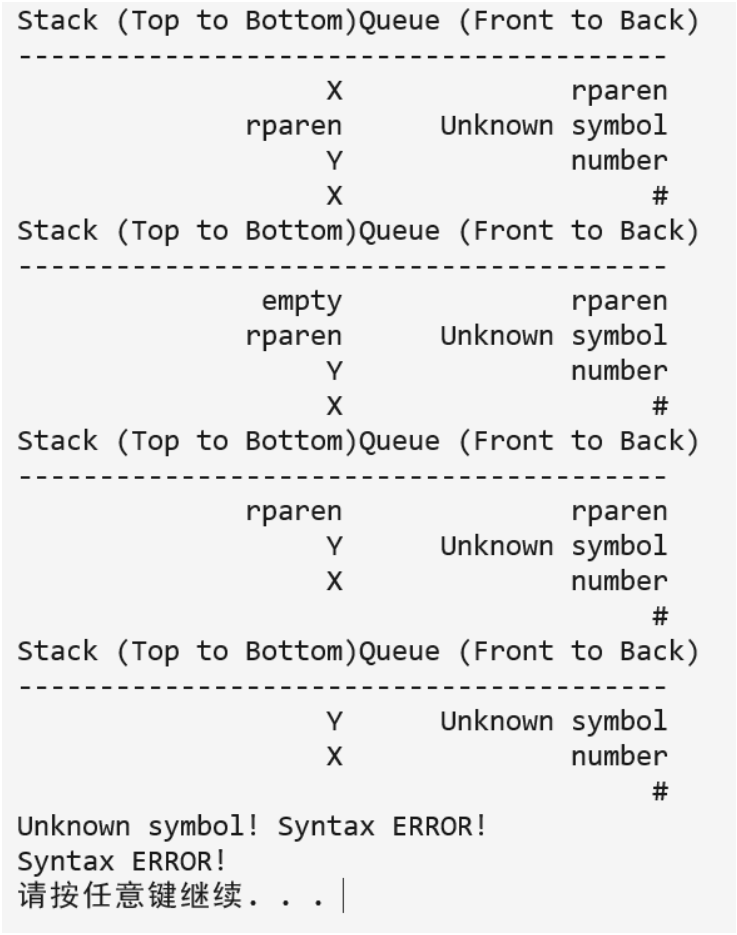

图 4　反例输出结果

3.5. 反例 4——括号内无内容

输入如下，出现了空括号当作表达式的情况。

```
1. -1+2+()+3*(7/(5+6)-8)
```

输出。

```
(minus, -)
(number, 1)
(plus, +)
(number, 2)
(plus, +)
(lparen, ( )
(rparen, ))
(plus, +)
(number, 3)
(times, *)
(lparen, ( )
(number, 7)
(slash, /)
(lparen, ( )
(number, 5)
(plus, +)
(number, 6)
(rparen, ))
(minus, -)
(number, 8)
(rparen, ))
```

```
                                        #
Stack (Top to Bottom)Queue (Front to Back)
----------------------------------------
              lparen              lparen
                   S              rparen
              rparen                plus
                   Y              number
                   X               times
                                  lparen
                                  number
                                   slash
                                  lparen
                                  number
                                    plus
                                  number
                                  rparen
                                   minus
                                  number
                                  rparen
                                       #
Stack (Top to Bottom)Queue (Front to Back)
----------------------------------------
                   S              rparen
              rparen                plus
                   Y              number
                   X               times
                                  lparen
                                  number
                                   slash
                                  lparen
                                  number
                                    plus
                                  number
                                  rparen
                                   minus
                                  number
                                  rparen
                                       #
Syntax ERROR!
Excepted S at position 6, but got rparen
Syntax ERROR!
请按任意键继续. . .
```

图 5　反例输出结果

四、 实验总结

本次实验的主要任务是实现一个基于 LL(1)分析法的语法分析器。通过对输入表达式进行语法分析，我们能够判断其是否符合 PL/0 语言的语法规范。实验中，我们使用了 LL(1)表驱动分析法，并实现了两个关键函数 `match_S` 和 `terminal_match`，它们分别负责处理非终结符和终结符的匹配过程。

通过对不同类型的输入表达式进行测试，我们验证了语法分析器的正确性和健壮性。实验结果表明，语法分析器能够正确识别合法表达式，并能准确地报告语法错误，包括未定义符号、括号不匹配、未知符号和空括号等情况。

在实验过程中，我们还遇到了一些挑战，例如如何设计一个高效的 LL(1)分析表，以及如何处理各种边界情况。虽然实现过程并不顺利，但我们成功地克服了这些困难，并在实践中加深了对 LL(1)分析法的理解。 通过本次实验，我们不仅掌握了 LL(1)分析法的基本原理和实现方法，还提高了我们在编程和调试方面的能力。我们认识到，语法分析是编译器设计中的一个重要环节，它为后续的语义分析和代码生成奠定了基础。通过对语法分析器的实现，我们对编译原理有了更深入的理解，也为今后的学习和研究打下了良好的基础。