编译原理

实验（一）表达式翻译器

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2023年3月17日
学　　院	信息学院
课程名称	编译原理

实验（一）表达式翻译器

目录

1 选题 **1**

2 实验目的 **1**

3 实现思路 **1**

4 问题**1**、如何使词法分析和语法分析的实现在形式上独立？ **1**

5 问题**2**、这样实现符合“语法制导翻译方案”吗？ **2**

1 选题

在程序pInOrderToPost3.cpp的基础上，完成“二、实验内容”中的要求。

2 实验目的

构造一个中缀表达式到后缀表达式的翻译器，初步了解递归下降语法分析原理及语法制导翻译的过程。

3 实现思路

首先，为了支持变量、乘除法和括号，我们需要对文法进行修改，如公式(1)。其中，`print`函数用于输出后缀表达式，`identifer`表示变量名。

$$\begin{aligned}
expr \rightarrow & expr + term \ \{print('+')\} \\
| & expr - term \ \{print('-')\} \\
| & term \\
term \rightarrow & term * factor \ \{print('*')\} \\
| & term/factor \ \{print('/')\} \\
| & factor \\
factor \rightarrow & (expr) | item \\
item \rightarrow & 0 \ \{print('0')\} \\
| & 1 \ \{print('1')\} \\
| & \dots \\
| & 9 \ \{print('9')\} \\
| & identifer \ \{print(lexeme)\}
\end{aligned} \tag{1}$$

其中，`identifer`的定义如(2)。

$$identifer := [a - zA - Z][a - zA - Z0 - 9]* \tag{2}$$

然后使用“语法制导定义”方法进行实现。

4 问题1、如何使词法分析和语法分析的实现在形式上独立？

为了在满足“按需词法分析”的要求下，使词法分析和语法分析的实现在形式上独立，我使用了C++的协程（coroutine）技术[1]。协程是一种程序组织方式，它允许程序在执行过程中暂停，然后在稍后的某个时间点恢复执行。节选的一段代码如代码 1。当使用`co_yield`时，程序会保留状态暂停执行，并返回指定的值。

程序返回的`generator<token>`对象可以被视为一个迭代器，可以使用`for`循环遍历。遍历时每当自增迭代器时，程序会从上次暂停并返回的位置之后恢复执行，直到遇到下一个`co_yield`语句暂停执行，或者遇到`co_return`结束执行。

因此，在调用`scan`函数时，虽然会立即返回，但是在遍历返回的`generator<token>`对象时真正的求值才会发生。借助这一特性，可以在代码形式上使得词法分析和语法分析的实现独立。然而在执行逻辑上，词法分析仍是由语法分析驱动的。这样实现能够在满足模块间“高内聚、低耦合”的要求下，依然保持了如同pInOrderToPost3.cpp中一样的执行逻辑。

[1]编译使用C++协程的程序需要编译器支持C++20，需要使用VS2019以上的版本

代码 1 使用协程实现词法分析

```
std::experimental::generator<token> scan(std::string code)
{
        auto it = code.begin();
        while (it != code.end())
        {
                if (*it == ' ' || *it == '\t' || *it == '\n' || *it == '\r')
                {
                        ++it;
                }
                else if (std::isdigit(*it))
                {
                        auto peek = it;
                        while (std::isdigit(*peek))
                        {
                                ++peek;
                        }
                        token t{};
                        t.type = token_type::NUM;
                        t.lexeme = std::string(it, peek);
                        it = peek;
                        co_yield t;
                }
                ...
                else
                {
                        throw std::runtime_error("Invalid token");
                }
        }
        co_return;
}
```

5 问题2、这样实现符合“语法制导翻译方案”吗？

是的。

根据问题1的解释，当调用函数scan时，实际上的词法分析并没有发生。在语法分析的过程中，当需要一个token时，在调用所返回的generator的迭代器的operator++函数时，才会发生一次求值，进而获取下一个token。因此，虽然在代码的形式上类似于“进行完整的词法分析并返回了所有token”，但实际上只是在语法分析的过程中进行了“按需词法分析”，得到一个token。符合“语法制导翻译方案”。因为在执行逻辑上，词法分析仍是由语法分析驱动的。