编译原理汇报

由于ppt篇幅有限，在展示代码的同时不宜展示过多文字，这是ppt对应的文字版。

输出语句 print

`print`语句的实现位于`statement`函数中.

```
else if (sym == SYM_PRINT) {
     ...
    if (sym == SYM_LPAREN) {
         ...
        if (sym == SYM_RPAREN) {
            gen(PRT, 255, 0);
        } else {
            set1 = createset(SYM_RPAREN, SYM_COMMA, SYM_NULL);
            set = uniteset(set1, fsys);
            expression(set);
            gen(PRT, 0, 0);
            while (sym == SYM_COMMA) {
                getsym();
                expression(set);
                gen(PRT, 0, 0);
            }
            destroyset(set1);
            destroyset(set);
            if (sym == SYM_RPAREN) gen(PRT, 255, 0);
            else error(22); // Missing ')'.
        }
        getsym();
    }
    else error(33); // Missing '('.
}
```

当遇到`SYM_PRINT`时，首先检查是否有左括号`SYM_LPAREN`，然后进入一个循环，处理每个打印项。这通过`expression`函数求值表达式实现，并通过`gen`函数生成打印指令（`PRT`）。如果遇到逗号`SYM_COMMA`，则继续读取和处理下一个表达式。最后，检查是否有右括号`SYM_RPAREN`。这样，`print`语句就能够处理和打印一个或多个表达式的值。

添加作用域算符 ::

类似于C++的作用域运算符 `::` 的实现，确实需要与标识符（如常量名、变量名、函数名、指针名、数组名）紧密结合。在 `factor` 函数中，当解析到 `SYM_DOMAIN` 时，表明下一个符号应该是一个标识符。这时，编译器会查找这个标识符在全局作用域的符号表中的位置。如果找到，就生成一个加载指令（LOD），以获取该变量的值。这种方式允许访问在当前局部作用域之外声明的全局变量，模拟了C++中作用域运算符的行为。

实现步骤:

1. **词法分析：** 标记 `::` 为特定符号（如 `SYM_DOMAIN`）。
2. **语法分析：** 在 `factor` 函数中解析 `::`，确定其在表达式中的位置。
3. **语义检查：** 检查 `::` 后的标识符的有效性和作用域访问权限。
4. **代码生成：** 生成访问外部作用域变量的指令。
5. **解释执行：** 执行生成的代码，实现作用域跨越的变量访问。

扩展

实现作用域运算符 `::` 还有引用函数的功能，主要涉及到在函数调用时正确解析和定位全局作用域中的函数。在解析到函数调用时，如果遇到 `::` 运算符，编译器会在全局作用域的符号表中查找相应的函数标识符。找到后，会生成相应的调用指令（例如CALL指令），以调用全局作用域中定义的函数。这样，即使在嵌套的局部作用域中，也能够正确访问和执行全局作用域中的函数。