中国科学技术大学
University of Science and Technology of China

编译原理实验汇报

报告人：张芷苒

第 8 组 付斯珂 金旭童 张芷苒

2023 年 12 月 25 日

1 实验内容

2 实现过程

3 实验分工

4 实验总结

基于 2017 版本的 pl/0：

- 实验 1 完成了指针和数组的变量声明、指针变量赋值和访问、数组元素的读写，以及输出语句 print
- 实验 2 扩展 PL/0 语言，添加类似于 C++ 语言的作用域算符:: 用来访问在过程（procedure）嵌套声明中的外层过程所声明的同名变量。完成相关的词法、语法、语义检查、代码生成及解释执行等编译实现。

添加 symtype:

```
SYM_PRINT,  // print
SYM_LBRACK, // [
SYM_RBRACK, // ]
SYM_AMPERSAND, // &
SYM_DOMAIN//lab2::
```

添加标识符: ID_ARRAY, ID_POINTER
添加指令: PRT, LODA, LEA, STOA

修改符号表结构体，增加 dimension 数组和 star:

```
typedef struct
{
    char  name[MAXIDLEN + 1];
    int   kind;
    short level;
    short address;
    int  dimension[MAX_ARRAY_DIM];
    int  star;//指针*嵌套层数

} mask;
```

用于处理变量和过程的声明。它检查 SYM_CONST、SYM_VAR、SYM_PROCEDURE 符号，然后根据不同的符号进行相应的处理，如常量声明、变量声明、过程声明。

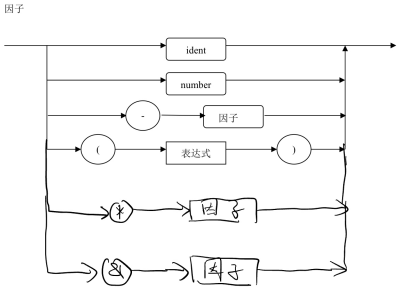
- 指针声明：通过读取 var 关键字后面的 * 个数，并将这个信息记录在符号表中。
- 数组声明：通过 dim_declaration 函数读取数组的各个维度信息。

处理程序中的语句，如赋值、条件判断、循环等。它首先检查当前的符号（如 SYM_IF、SYM_WHILE、SYM_CALL 等），然后根据不同的符号执行不同的语句处理逻辑。

- 指针赋值：根据变量名 id 前的 * 个数来决定生成加载地址 LODA 指令的次数。
- 数组赋值：通过 dim_position 函数解析标识符后的多个 [...]...[...] 来确定数组元素的具体赋值位置。

‘expression‘函数用于计算表达式的值，它处理加法和减法运算符，并能递归地处理更复杂的表达式。
‘term‘函数则处理乘法和除法运算符，它在 ‘expression‘的基础上进一步细分表达式的组成部分。
这两个函数在语法分析阶段被调用，负责解析源代码中的算术表达式，并生成相应的中间代码或机器代码，以便后续的编译步骤可以使用。未作较大改动。

‘factor‘函数的功能被扩展以处理 ‘&‘（取地址）和 ‘*‘（解引用）操作。当 ‘factor‘遇到这些符号时，它会相应地生成指令来处理地址和指针的操作。
‘factor‘函数的核心作用是确定因子的值，并将这个值置于运行时的栈顶，这对于后续的操作和表达式的计算至关重要。



‘print‘语句的实现位于 ‘statement‘函数中。

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‘。

```
else if (sym == SYM_PRINT) {
    getsym();
    if (sym == SYM_LPAREN) {
        getsym();
        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 '('.
```

当遇到作用域运算符‘::‘时，首先将作用域设置为全局作用域，然后查找符号表中的标识符。如果找到对应的变量，会生成一个加载指令（LOD），以获取该变量的值。这允许访问在当前作用域之外声明的变量，实现了类似于 C++ 中作用域运算符的功能。

```
if (sym == SYM_DOMAIN) {
    // 将当前作用域设置为全局作用域
    getsym();
    if (sym == SYM_IDENTIFIER) {
        // 查找标识符在符号表中的位置
        if (!(i = position(id))) {
            error(11); // 未声明的标识符
        } else {
            mk = (mask*)&table[i];
            if (mk→kind == ID_VARIABLE) {
                // 生成加载指令
                gen(LOD, level - mk→level, mk→address);
            }
            ...
        }
    }
    ...
}
```

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。
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。

付斯珂

- 添加指针和数组的扩展（实验 1 部分）
- 安排小组分工

金旭童

- 添加作用域算符的扩展（实验 2 部分）
- 负责合并两个实验代码

张芷苒

- 协助代码实现
- 代码 debug，添加注释
- 负责汇报总结文件和汇报展示

结论

在本次实验中，我们完成了以下基于 pl/0 的扩展：

- 指针和数组的变量声明、赋值和访问；数组元素读写的一般形式、指针化表示；
- 输出语句 printf 的实现；
- 添加作用域算符::；完成相关的词法、语法、语义检查、代码生成及解释执行等编译实现；
- 所有三个测试样例都能通过。

谢谢! 下面是我们的实验展示过程。