编译原理实验报告

实验三：中间代码生成

学院：	计算学部	指导老师：	单丽莉
学号：	2021112845	姓名：	张智雄

一、实验目的

1. 巩固对中间代码生成的基本功能和原理的认识。
2. 能够基于语法指导翻译的知识进行中间代码生成。
3. 掌握类高级语言中基本语句所对应的语义动作。

二、实验内容

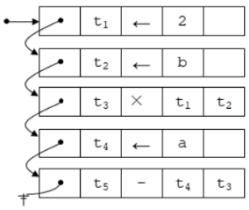
在词法分析、 语法分析和语义分析程序的基础上， 将 C--源代码翻译为中间代码（ 三地址代码形式）。

2.1 实验环境

Ubuntu 22.04；GCC version 11.4.0；GNU Flex version 2.6.4；GUN Bison version 3.8.2

2.2 实验过程

2.2.1 中间代码的表示（线形）



本实验基于**双向循环链表**来保存生成好的中间代码。链表以增加实现的复杂性为代价换得了极大的灵活性，可以进行高效的插入、删除以及调换位置操作，并且几乎不存在代码最大行数的限制。对于此数据结构的具体解释如下：

- **Operand（操作数）：**用于表示中间代码中的操作数，包括常量、变量、地址、标签、函数和关系运算。Operand 结构体**包含一个枚举类型，表示操作数的种类，以及一个联合体**，存储操作数的值或标识符名称。
- **InterCode 和 InterCodes（中间代码）：**InterCode 用于表示生成的中间代码的单条指令。包含**一个枚举类型，表示指令的种类，以及一个联合体**，存储指令如一元操作、赋值操作、二元操作、条件跳转、数组声明；InterCodes 用于表示生成的中间代码的双向链表。
- **Arg 和 ArgList（参数列表）：**Arg 用于表示中间代码中的参数列表。每**个节点包含一个操作数指针，以及指向下一个参数的指针**；ArgList 用于表示参数列表，其中包含**指向参数链表头部和当前节点的指针**。
- **InterCodeList（**中间代码列表头**）：**表示生成的中间代码列表的**头结点和当前节点，以及临时变量 tempVarNum 和标签 labelNum 的编号**，tempVarNum 和 labelNum 用于生成临时变量和标号，每当生成一个变量或标号时，这两变量均自增 1，以确保变量名不重复。

2.2.2 翻译模式

本实验主要是为每个主要的语法单元“X”都设计相应的翻译函数“translate_X”，对语法树的遍历过程也就是这些函数之间互相调用的过程。每种特定的语法结构都对应了固定模式的翻译“模板”，最终将其转换为三地址指令的中间代码。

具体来说，该生成器可以实现**函数定义和调用、加减乘除运算、一维数组变量的创建和使用、if/if-else、while 等语句**的中间代码生成。

2.2.2.1 函数定义与调用

当遇到**函数定义**时，会生成对应的中间代码 IR_FUNCTION，其中操作数为函数的名称。

- 函数定义出现全局作用域内，对应产生式为 FunDec -> ID LP VarList RP| | ID LP RP。针对有形参列表的函数定义，还需要遍历 VarList 从而生成函数参数声明 IR_PARAM 的中间代码。由于没有全局变量使用，ExtDecList 不涉及中间代码生成，类型声明也不涉及，所以只需要处理 FunDec 和 CompSt。
- 同时函数需要生成返回值传递 IR_RETURN 代码。return 语句对应产生式为 Stmt -> RETURN Exp SEMI。使用 translateExp 函数翻译 Exp 并将结果存储到临时变量“t+tempVarNum” 中，调用 genInterCode 函数生成 RETURN 语句的中间代码。

当在代码中遇到**函数调用**语句时，对应的产生式为 Exp -> ID LP Args RP | | ID LP RP，需要产生对应的中间代码，形式为 x := CALL f。对于函数调用的参数传递，需要扫描参数列表 Args，并生成相应的 IR_ARG 中间代码。

2.2.2.2 基本表达式

这类语句的处理集中在 translateExp 函数中，该函数囊括了 IR_ASSIGN、IR_ADD、IR_SUB、IR_MUL、IR_DIV 等中间代码生成。首先判断当前节点的类型，

- 如果是整数常量或标识符，则直接为其生成对应的操作数。
- 如果是表达式，则需要递归地处理左右子表达式，为它们分配临时变量，并生成对应的中间代码。根据操作符的类型，生成相应的中间代码，将操作数和结果存储位置传递给 genInterCode 函数。

2.2.2.3 条件语句 if/if-else

对于 if 语句中的条件表达式，通过 translateCond 函数将其翻译成中间代码。具体而言，此函数根据条件表达式的结构，生成相应的中间代码（包括条件跳转 IR_IF_GOTO 和跳转指令 IR_GOTO），支持条件表达式中的关系运算符（如大于、小于等）和逻辑运算符（如与、或、非），并将条件表达式的真假情况映射到标签（label）上，便于后续的条件跳转。同时此函数还使用了**基于逻辑运算的短路特性的短路翻译**：

- 当遇到“与”运算时，如果第一个操作数为 ture，则直接跳转到 labelFalse 标号处，必须所有的操作数都为 true 时，才能跳转到 labelTrue 标号处；
- 当遇到“或”运算时，如果第一个操作数为 true，则直接 labelTrue，否则继续操作数的运算，当所有的操作数都为 false 时，才跳转到 labelFalse 处。

生成标签（label）后，还需要根据产生式对 labelFalse 和 labelTrue 进行**回填**：

- 当产生式为 Stmt -> IF LP Exp RP Stmt 时，当开始处理 Stmt 之前，需要回填 labelTrue，处理完 Stmt 后，回填 labelFalse。
- 当产生式为 Stmt -> IF LP Exp RP Stmt ELSE Stmt，labelFalse 成为跳转到 else 语句块的位置标号，此外还需要在 if 语句块的末尾添加 labelnext 标号来无条件跳出 if-else 语句。当处理完 ELSE Stmt 后对 labelnext 进行回填。

2.2.2.4 循环语句 while

while 语句对应语法树的产生式为 Stmt -> WHILE LP Exp RP Stmt。在进入 while 循环之前，创建两个标签，一个用于循环体的起始位置，另一个用于循环体之后的位置。

而后翻译条件表达式 Exp，并在条件为假时跳转到循环体之后的位置。接着，生成循环体的中间代码，包括循环体内部的语句。在循环体执行完毕后，生成无条件跳转指令，使程序跳转到循环体的起始位置。最后，回填标签完成循环控制的逻辑。

2.2.2.5 一维数组变量

遇到一维数组**定义**(VarDec -> | VarDec LB INT RB)时，程序调用 genInterCode(IR_DEC, ...)来生成中间代码。该中间代码表示为数组分配内存空间，其中包括数组的名称和大小。若试探性地扫描到产生式右部 VarDec 的后继产生式依旧是一个数组定义的形式，那么说明这个数组是高维数组，程序报错。

当程序需要**访问**一维数组的元素(Exp -> Exp LB Exp RB)时，程序会生成对应的中间代码来计算数组元素的偏移量，并将数组基地址与偏移量相加，从而得到要访问的元素的地址。在生成访问一维数组元素的中间代码时，程序还会考虑是否需要对数组进行取址操作。如果数组是一个表达式的一部分，需要使用数组的地址，程序会生成对应的取址操作的中间代码。

2.2.2.6 对高维数组和结构体的报错

当遇到定义数组的产生式(VarDec -> | VarDec LB INT RB)时，程序试探性地扫描产生式右部 VarDec 的后继产生式是否依旧是数组定义的形式，若是则说明为高维数组，程序报错。

由于程序中在全局定义的产生式中直接忽略了结构体的定义，所以使用结构体的报错信息发生在语法树中存在 Exp -> Exp DOT ID 或检测到某个变量的类型是结构体时进行报错。

以上任意两种错误发生时，都会将 interError 置为 True，从而停止中间代码的生成。

2.4 实验结果

在 linux 环境下，在文件夹目录下中运行 make 指令编译所有的文件，而后输入 make test 指令便可以测试所有的样例。测试结果如下，能够正确分析所有必做用例：

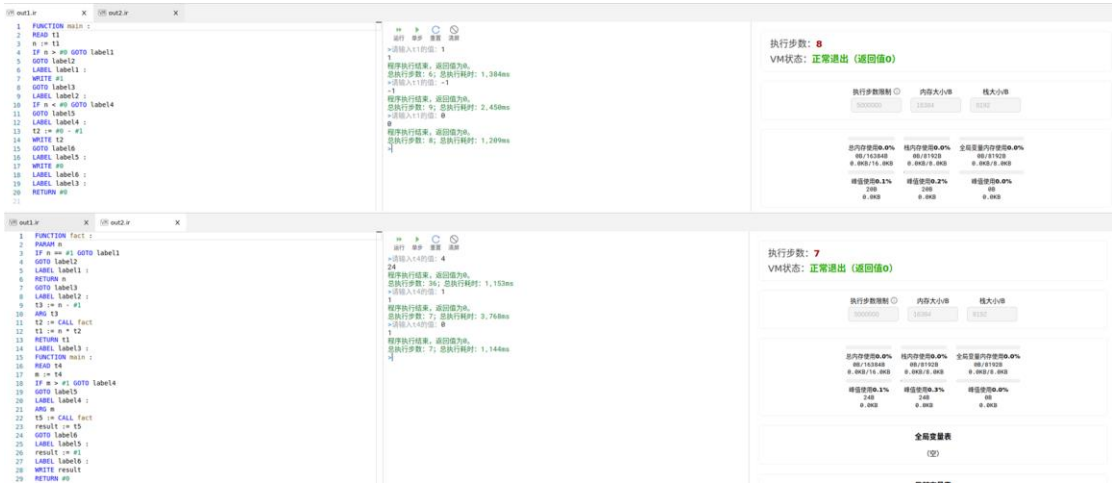


图 1 测试结果

三、实验收获

本次实验深入学习了中间代码生成的基本原理和实践操作。通过词法、语法和语义分析，成功将 C--源代码转换为中间代码，掌握了编译器关键步骤。进一步熟悉了函数定义与调用、条件语句等核心语法结构的中间代码生成，并对循环语句和数组处理有了更深入的理解。