编译原理实验三：中间代码生成

——杨嘉兴+21311054+2807593076@qq.com

一、 底层数据结构

本次实验的大部分代码老师均已给出，在中间代码生成的代码实现中，采用的底层数据包括**操作数结构体**和**中间代码结构体**，而且中间代码的表示主要采用的是**线性的双向循环链表**，大大提高了增删改的灵活性。

1. 操作数结构体

该操作数结构体表示了用于**中间代码的操作数**，包含了中间代码的**类型**和**具体值**，以及一个**存放数组**或**结构体类型变量的类型**，还有将该操作数与其它操作数进行链接的**指针**。其中操作数的类型主要包括**变量、临时变量、常量、标签、函数、取地址操作数、解引用操作数**，而存储在操作数中具体的值主要包括**临时变量的编号、标记的编号、常量的值、函数**以及**变量的名字、取地址**和**解引用指向的操作数**。

2. 中间代码结构体

该中间代码结构体主要包含了中间代码的**类型、操作数、附加信息**和**链表指针信息**等。其中中间代码的类型主要包括**标签、函数、赋值、加法、减法、乘法、除法、存储到内存、无条件跳转、条件跳转、返回、变量声明、参数、函数调用、函数参数、读、写、空指令**等，而附加信息主要包括了**条件跳转的关系运算符**和**条件跳转变量的大小**。而在链表指针信息中，主要包括了**前一个中间代码**和**后一个中间代码**的指针信息，因为我们采用的中间代码表示形式为**双向循环链表**。

二、 功能实现

本次实验完成了所有的实验内容，主要包括以下的**三个要求：**

1. 修改了前面对 C--源代码的假设 2 和 3，使源代码中可以出现结构体类型的变量（但不会有结构体变量之间直接赋值），而且结构体类型的变量可以作为函数的参数（但函数不会返回结构体类型的值） 。

2. 修改前面对 C--源代码的假设 2 和 3，使源代码中一维数组类型的变量可以作为函数参数（但函数不会返回一维数组类型的值），而且可以出现高维数组类型的变量（但高维数组类型的变量不会作为函数的参数或返回类值）。

3. 考虑了优化生成的中间代码的**执行效率**。

三、 挖空部分的代码填充

1. 优化加法：

我们需要考虑如何对加法运算的中间代码生成进行优化，我们首先需要考虑加法的一些特殊情况，比如当两个操作数都是常量、其中一个操作数为常量而且为 **0** 而且另一个操作数不为获取地址操作数类型或获取值操作数类型时，我们可以进行优化。但是当这些情况都不存在时，我们就需要生成实际的加法指令。首先，使用 **malloc()**函数为一个名为 **code1** 的 **InterCode** 结构体分配内存空间，将该结构体的类型设置为 **PLUS_IR**，表示**加法指令**类型。接着，将目标操作数 **dest** 存储在结构体的第一个操作数位置，将源操作数 **src1** 存储在第二个操作数位置，将源操作数 **src2** 存储在第三个操作数位置。最后，返回创建的加法指令 **code1**。

2. 优化乘法：

我们需要考虑如何对乘法运算的中间代码生成进行优化，我们首先需要考虑乘法的一些特殊情况，比如当两个操作数都是常量、当其中一个操作数为常量而且为 **0**、当其中一个操作数为常量而且为 **1** 而且另一个操作数不是获取地址操作数类型或获取值操作数类型时，我们可以进行优化。但是当这些情况都不存在时，我们就需要生成实际的乘法指令了。首先，使用 **malloc()**函数为一个名为 **code1** 的 **InterCode** 结构体分配内存空间。然后，将该结构体的类型设置为 **MUL_IR**，表示乘法指令类型。接着，将目标操作数 **dest** 存储在结构体的第一个操作数位置，将源操作数 **src1** 存储在第二个操作数位置，将源操作数 **src2** 存储在第三个操作数位置。最后，返回创建的乘法指令 **code1**。

3. 赋值表达式中单个变量作为左值：

首先，我们通过调用 **findSymbolAll()**函数找到该变量的符号表项，并使用 **getVar()**函数获取该变量的操作数形式。然后创建一个临时变量 **tmp1**，用于存储右侧表达式的计算结果，随后调用 **translateExp()**函数将右侧的表达式翻译为中间代码，并将结果存储在临时变量 **tmp1** 中，而且生成的中间代码存储在 **code1** 中。然后使用 **malloc()**函数为一个名为 **code2** 的 **InterCode** 结构体分配内存空间，将 **code2** 的类型设置为 **ASSIGN_IR**，表示赋值指令类型。将左值变量 **var** 存储在 **code2** 的第一个操作数位置。将右侧表达式的计算结果 **tmp1** 存储在 **code2** 的第二个操作数位置。调用 **insertInterCode()**函数将 **code2** 插入到 **code1** 之前，实现将赋值指令插入到表达式计算指令之前。如果存在 **place**，即需要将计算结果存储到指定的位置，使用 **operandCpy()**函数将左值变量 **var** 拷贝到 **place** 中。最后返回生成的中间代码 **code1**。

4. 条件表达式：

首先，我们创建两个标签 **label1** 和 **label2** 用于条件跳转，随后使用 **malloc()**函数为一个名为 **code1** 的 **InterCode** 结构体分配内存空间，将 **code1** 的类型设置为 **ASSIGN_IR**，表示赋值指令类型，同时将目标操作数 **place** 存储在 **code1** 的第一个操作数位置，将常数值为 **0** 的操作数存储在 **code1** 的第二个操作数位置。然后调用 **translateCond()**函数翻译条件表达式，并将生成的中间代码存储在 **code2** 中。随后使用 **malloc()**函数为一个名为 **code3** 的 **InterCode**

结构体分配内存空间，并将其类型设置为 **LABEL_IR**，表示标签指令类型，并且将 **label1** 存储在 **code3** 的操作数位置。然后使用 **malloc()**函数为一个名为 **code4** 的 **InterCode** 结构体分配内存空间，并将其类型设置为 **ASSIGN_IR**，表示赋值指令类型，将目标操作数 **place** 存储在 **code4** 的第一个操作数位置，将常数值为 **1** 的操作数存储在 **code4** 的第二个操作数位置。随后使用 **malloc()**函数为一个名为 **code5** 的 **InterCode** 结构体分配内存空间，并将其类型设置为 **LABEL_IR**，表示标签指令类型，将 **label2** 存储在 **code5** 的操作数位置。然后我们调用 **insertInterCode()**函数分别将 **code2**、**code3**、**code4**、**code5** 插入到 **code1** 之前。这样可以保证生成的中间代码的顺序正确。最后我们返回生成的中间代码 **code1** 即可。

5. **条件表达式的翻译模式：**

- **如果条件表达式是关系运算符（Relop）**，则创建临时变量 **tmp1** 和 **tmp2**，并调用 **translateExp()**函数分别翻译左右子表达式，生成相应的中间代码。然后创建一个条件跳转的中间代码（**IF_GOTO_IR**），设置操作数为 **tmp1**、**tmp2** 和 **labelTrue**，表示如果条件成立，则跳转到 **labelTrue**。同时创建一个无条件跳转的中间代码（**GOTO_IR**），设置跳转目标为 **labelFalse**，表示条件不成立时跳转到 **labelFalse**。将生成的中间代码插入到 **code1** 中，并返回 **code1**。
- **如果条件表达式是逻辑非（NOT）**，则递归调用 **translateCond()**函数翻译子表达式，并交换 **labelTrue** 和 **labelFalse** 的位置，表示条件取反。返回子表达式的翻译结果。
- **如果条件表达式是逻辑与（AND）**，则创建一个标签 **label1**，并分别调用 **translateCond()** 函数翻译左右子表达式。生成一个标签中间代码（**LABEL_IR**），设置操作数为 **label1**，表示标记 **label1** 的位置。将生成的中间代码插入到 **code1** 中，并返回 **code1**。
- **如果条件表达式是逻辑或（OR）**，则创建一个标签 **label1**，并分别调用 **translateCond()** 函数翻译左右子表达式。生成一个标签中间代码（**LABEL_IR**），设置操作数为 **label1**，表示标记 **label1** 的位置。将生成的中间代码插入到 **code1** 中，并返回 **code1**。
- **对于其他情况，**即条件表达式是其他类型的表达式时，创建临时变量 **tmp1**，并调用 **translateExp()**函数翻译整个条件表达式，生成相应的中间代码。然后创建一个条件跳转的中间代码（**IF_GOTO_IR**），设置操作数为 **tmp1**、常数值为 **0** 的操作数和 **labelTrue**，表示如果条件成立，则跳转到 **labelTrue**。同时创建一个无条件跳转的中间代码（**GOTO_IR**），设置跳转目标为 **labelFalse**，表示条件不成立时跳转到 **labelFalse**。将生成的中间代码插入到 **code1** 中，并返回 **code1**。

四、 代码编译与运行

1. **代码编译：**进入 **Code** 目录下执行 **make** 命令，会在该目录下产生相应的可执行文件 **parser**。

2. **代码测试：**编译得到 **parser** 文件后，通过**./parser ../Test/test_d1.cmm ../result/test_d1.ir** 对相应的测试文件进行测试，随后可在 **result** 文件夹中得到该测试文件对应的中间代码生成。