编译原理 实验 3: 中间代码生成

冯诗伟 161220039

1 实现功能

实现了中间代码的生成，但是选作任务 (结构体相关) 有些 bug 没有完全处理掉，可以检查出高维数组的出现和数组作为参数的情况。

2 如何编译

由于改成了 C++ 代码，所以也擅自修改了 Makefile，里面使用了 g++。MakefileOld 是助教之前给的文件，Makefile 是用来编译 C/C++ 的新文件。助教只需要修改 Makefile 文件中 test 的部分即可。

3 (自认为的) 程序 Feature

3.1 Visitor 模式

由于 Visitor 模式适合处理这种对象类型很多但变化较少、对象上的操作类型很多的情况，在实验二语义分析部分就把 C 实现的语法树转化为 C++ 实现的 AST，使得对于 AST 的操作相对集中。总体逻辑如下：

```
/*  Code/ast/Transform.cpp  */
Program *transToAST(MultiNode *root){
    //从C的语法树转为C++的AST, program指向AST根结点
    Program *program = transProgram(root);

    //TypeVisitor负责建立符号表
    TypeVisitor typeVisitor;
    program->accept(typeVisitor);

    //IRBuilderVisitor负责生成中间代码
    IRBuilderVisitor irBuilderVisitor;
    irBuilderVisitor.symbolTable = typeVisitor.symbolTable;
    irBuilderVisitor.funTable    = typeVisitor.funTable;
    program->accept(irBuilderVisitor);
    irBuilderVisitor.printIRList(); //输出中间代码
    return program;
}
```

3.2 IR 的数据结构

本次实验选用了线型中间代码。相关的部分都在 Code/ast/ir 文件夹下：

- IRInstruction.h: 定义了一条中间代码的结构。由于中间代码形式不同，src1、src2、dest 不总是全都被使用到，总共可以分为三类，使用三种不同的构造函数。(src1, src1+dest, src1+src2+dest)。

```
struct IRInstruction{
    IROperator op;
    string src1;
    string src2;
    string dest;
    virtual ~IRInstruction() {}
};
```

- IRInstruction.cpp: 实现中间代码的打印，即如何从数据结果打印到文件中。

```
string IRInstruction::toString() {
    string str;
    switch (this->op) {
        // src1
        case IR_LABEL: str = "LABEL " + src1 + " :"; break;
        ......
        //src1 dest
        case IR_DEC: str = "DEC " + dest + " " + src1; break;
        ......
        //src1 src2 dest
        case IR_PLUS: str = dest + " := " + src1 + " + " + src2; break;
        ......
        default: cerr << "Invalid IR operator" << endl; break;
    }
    return str;
}
```

- IRBuilderVisitor.h IRBuilderVisitor.cpp 是中间代码生成最核心的部分。

3.3 IRBuilderVisitor 的实现

IRBuilderVisitor 的实现是在 IRBuilderVisitor.cpp 中，大体的思想都参照实验讲义的翻译方法。但是还有些不同之处需要自己来解决。

比如讲义中的 translate 函数均有 3-4 个参数，其中第一个参数对应于 visit 的参数，最后的符号表是 visitor 的成员变量，其他的参数就需要新建一个成员变量，每一次调用 visit 函数前都对它进行赋值。

再比如，讲义中的 translate 函数有的情况下会调用不同的 translate 函数，比如 if 和 while 语句的翻译都会同时调用 translate_Cond 和 translate_Stmt 函数，这对于依靠递归调用的 visitor 函数来说无法辨别和

实现，为此我额外写了一个 translate_Cond 函数，供需要的时候调用。

4 可改进之处

- 由于 visit 函数只有一个参数，所以有些时候不得不借助外部力量 (比如成员变量) 来达到目的，代码没有那么优雅:)。
- 结构体相关的内容写了一部分，比如可以得到结构体内部成员的偏移量，结构体成员左值赋值，但是嵌套访问、结构体数组访问的时候有些 bug 没有及时解决，比较遗憾。