Lab2: 中间代码生成

> Author wzj Data: 2025.5.11

实现功能

语法制导的代码生成

参考实验手册， 表达式代码生成， 语句代码生成， 条件判断语句代码生成， 函数代码生成都给出了具体的生成规则，较为简单，以 IfStmt 的 Translate function 为例， 按照实验报告中的规则，先翻译条件表达式，在按序翻译语句，在它们之间插入Label即可

```
IR::Code IRTranslator::translateIfStmt(AST::IfStmtPtr node) {
  IR::Code ir;
  if (!node->false_stmt) {
    auto label_true = new_label();
    auto label_false = new_label();
    auto code1 = translateCond(node->cond, label_true, label_false);
    auto code2 = translate(node->true_stmt);
    // return code1 + [LABEL label1] + code2 + [LABEL label2]
    auto LT = IR::Label::create(label_true);
    auto LF = IR::Label::create(label_false);
    std::move(code1.begin(), code1.end(), std::back_inserter(ir));
    ir.push_back(LT);
    std::move(code2.begin(), code2.end(), std::back_inserter(ir));
    ir.push_back(LF);
  }

  else {
    auto label_1 = new_label();
    auto label_2 = new_label();
    auto label_3 = new_label();
    auto code1 = translateCond(node->cond, label_1, label_2);
    auto code2 = translate(node->true_stmt);
    auto code3 = translate(node->false_stmt);

    auto L1 = IR::Label::create(label_1);
    auto L2 = IR::Label::create(label_2);
    auto L3 = IR::Label::create(label_3);
    auto GOTO3 = IR::Goto::create(label_3);
    std::move(code1.begin(), code1.end(), std::back_inserter(ir));
    ir.push_back(L1);
    std::move(code2.begin(), code2.end(), std::back_inserter(ir));
    ir.push_back(GOTO3);
    ir.push_back(L2);
    std::move(code3.begin(), code3.end(), std::back_inserter(ir));
    ir.push_back(L3);
  }
```

```
    return ir;
  }
```

较为复杂的是全局变量和数组代码生成，实验报告中未给出具体生成规则，也是本实验报告中的技术亮点所在： 首先，因为这两种类型的变量实质上都是地址/指针，它们的使用和赋值是相似的，需要定义新的指令Global, DEC等

```
class Global;
using GlobalPtr = std::shared_ptr<Global>;
class Global : public Node {
 public:
  std::string name;
  int size;
  std::vector<int> values;

  Global(const std::string &name, int size, const std::vector<int> &values
= {})
      : name(name), size(size), values(values) {
    if (values.empty()) {
      this->values = std::vector<int>(size / 4, 0);
    }
    if (values.size() != size / 4) {
      // padding rest space with 0
      this->values.resize(size / 4, 0);
    }
  }

  static GlobalPtr create(const std::string &name, int size, const
std::vector<int> &values = {}) {
    return std::make_shared<Global>(name, size, values);
  }

  std::string to_string() const override {
    std::string values_str;
    if (!values.empty()) {
      values_str = " = " + std::accumulate(values.begin() + 1,
values.end(),
        "#" + std::to_string(values[0]),
        [](const std::string &a, int b) { return a + ", #" +
std::to_string(b); });
    }
    return "GLOBAL " + name + ": #" + std::to_string(size) + values_str;
  }
};
```

篇幅所限，接下来说说我认为实验中最难的一部分，即翻译初始化列表。
和Lab2相似，可能会出现嵌套的初始化列表，这里我延续了Lab2时的思路，使用递归的思想，在解释时进行赋值。由于篇幅原因和与Lab2重复原因不在此赘述，代码中有较为详尽的注释

Reference

1. 实验过程中使用cursor/Gemini, 用于分析一些报错信息以及查询知识空缺