Lab2: 语义分析

Author: wzj Date: 2025.4.13

实现功能

符号表的实现

参考实验手册，实现函数式风格的符号表，在`SymbolTable`类维护一个符号表的栈(vector), 使用 enter_scope/exit_scope 进行作用域的维护

```
class SymbolTable {
 public:
  ...
  /// @brief The current scope depth
  int scope_depth = -1;
  /// @brief functional implementation of the symbol table
  /// scopes.back() is the current scope
  std::vector<std::unordered_map<std::string, SymbolPtr>> scopes;

};
void SymbolTable::enter_scope() {
  scopes.emplace_back();
  ++ scope_depth;
}

void SymbolTable::exit_scope() {
  if (!scopes.empty()) {
    scopes.pop_back();
    -- scope_depth;
  }
}
```

类型检查实现

类型检查需要遍历语法树节点，因此要为lab1中定义的各个AST类型实现对应的check函数

技术亮点

本次实验中较为复杂的是实现数组初始化以及数组类型匹配检查部分

在实现数组初始化检查时，对于嵌套列表（InitList），我使用了递归检查的方法, 否则（为正常的标量），则继续进行检查；通过对`filled_elements`的计算，来确定子数组大小以及是否溢出。关键代码如下：

```
TypePtr TypeChecker::checkInitList(AST::InitListPtr node,
  ArrayTypePtr array_type) {
  int filled_elements = 0;
```

```
    int total_elements = 1;
    for (auto d : array_type->dims) {
      total_elements *= d;
    }
    auto elements = node->elements;
    ArrayTypePtr subarray_type = nullptr;
    int subarray_size;
    for (auto element : elements) {
      auto val = std::dynamic_pointer_cast<AST::InitVal>(element);
      if (auto n = std::dynamic_pointer_cast<AST::InitList>(val->inits[0])) {
        // 子数组
        for (int i = 1; i < array_type->dims.size(); i++) {
            subarray_type = nullptr;
            subarray_size = 1;
            for (int j = i; j < array_type->dims.size(); j++) {
              subarray_size *= array_type->dims[j];
            }
            if (filled_elements % subarray_size == 0) {
              // 检查子数组的维度和初始化表达式的维度是否匹配
              subarray_type = ArrayType::create(array_type->element_type,
                                                std::vector<int>
(array_type->dims.begin() + i,

array_type->dims.end()));
              break;
            }
          }
          if (subarray_type == nullptr) {
            ASSERT(false, "Subarray type mismatch at line " +
                              std::to_string(node->lineno));
          }
          checkInitList(n, subarray_type);
          filled_elements += subarray_size;
      }
      else {
        // 标量
        auto type = check(val->inits[0]);
        if (!type->equals(array_type->element_type)) {
          ASSERT(false, "Initialization type mismatch at line " +
                            std::to_string(node->lineno));
        }
        filled_elements++;
      }

      if (filled_elements > total_elements) {
        ASSERT(false, "Excess initializers at line " +
                          std::to_string(node->lineno));
      }
    }
    return nullptr;
}
```

我在实现数组类型匹配检查时，由于没有在lab1中记录`AST::Lval`为array的dim信息，出现了一些错误，重新完善后使其能正确记录Array, 使用如下的`equal`方法即可判断数组类型是否一致。

```
bool ArrayType::equals(const TypePtr& other) const {
  auto other_type = std::dynamic_pointer_cast<ArrayType>(other);
  if (!other_type) return false;
  if (!element_type->equals(other_type->element_type)) return false;
  if (dims.size() != other_type->dims.size()) return false;
  for (size_t i = 1; i < dims.size(); i++) {
    if (dims[i] != other_type->dims[i]) return false;
  }
  return true;
}
```

其他的类型检查较为简单，按照模板提示逐一实现， 不赘述。对于函数参数等较为复杂的检查，逐一对比输入参数与定义的参数即可。

Reference

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