编译原理研讨课实验 PR002 实验报告

一、任务说明

本次实验要求完成对 CACT 代码的语法分析。根据 CACT 语言的语法规范，在.g4 文件中添加必要的属性，完善对语法树的访问接口，对输入的程序进行语义分析，并给出相应的错误报告。

二、成员组成

吕恒磊，康家锐，王东宇。

三、实验设计

1. 设计思路

根据 CACT 语言的语法规则，逐一分析可能出现的语义错误，并推测需要用到的属性、语法树接口处理方法、符号表内容等。实现过程中先完成符号表的相关内容，再分工完成属性添加和语法树接口处理。在后续实验过程中遇到需要添加的符号表内容，则添加相应成员、方法等，逐步完善语义分析功能。

2. 实验实现

首先是符号表的实现。

```
struct VarInfo
{
    std::string name;
    int btype;
    int is_const;
    //int vartype;    // basic or array
    int array_len;  // if this variable is basic type, then it is set to 0
};
```

上图是我们定义的变量信息存储结构体，包含变量名、基础类型、是否为常量（常量信息也用同一结构体储存）、数组长度（非数组变量该值为 0）等信息。

```
struct ScopeTable
{
    int start_line;
    std::map<int, ScopeTable *> children;
    std::map<std::string, VarInfo> local_var;
    ScopeTable *parent;
    FuncInfo* curr_func;
};
```

上图是我们定义的作用域信息储存结构体，包含作用域开始行号（作为检索信息）、子作用域列表、局部变量信息表、父作用域指针、所属函数指针等信息。

```
struct FuncInfo
{
    std::string name;
    int ret_type;
    int cnt_para;
    std::map<int, VarInfo> para;
    ScopeTable scope;
};
```

上图是我们定义的函数信息储存结构体，包含函数名、函数返回值类型、参数个数、参数信息列表、函数作用域结构体等信息。

```
public:
    // global variable table
    std::map<std::string, VarInfo> global_var;
    // function table
    std::map<std::string, FuncInfo> func;
    // current scope
    ScopeTable* curr_scope;
```

上图是 SimbolTable 类的几个公共成员，分别是全局变量表、函数表和指向当前作用域的指针，其他的信息均会被添加到各自所属的函数信息结构体中。

```
//look up var in current scope
VarInfo * lookup_curr(const std::string &name);
//look up var in all scope
VarInfo * lookup_all(const std::string &name);
//look up function
FuncInfo * lookup_func(const std::string &name);
//look up parameter
VarInfo *lookup_para(FuncInfo * func, int index);

void addFunc(std::string name, int ret_type, int start_line);
void addPara(std::string name, int btype, int len);
void addVar(std::string name, int btype, int len, int is_const);
void addScope(int start_line);

void scope_ret();
```

上图是 SimbolTable 类的几个方法声明。lookup_curr 函数在当前作用域下查询变量，主要用于变量声明时的查重。lookup_all 函数从当前作用域开始逐级向上查询变量，主要用于使用变量时检索变量信息。lookup_func 函数查询函数，用于函数声明时的查重和函数调用时的检索函数信息。lookup_para 函数查询函数的参数，用于函数声明时的参数查重，以及函数内变量声明时的查重，以及函数调用时检索参数信息。后面的几个 add 函数均用于向符号表中添加相关信息，并在发生重复时发出错误报告。

其次是语法树属性添加和访问接口的处理。

我们主要添加的是数据类型属性，用来在不同节点间判定是否出现类型错误。也有下图所示的 in_loop 属性，用于标明当前处理语句是否在循环中，以确定能否使用 continue 语句和 break 语句。

```
stmt
    locals[
        int in_loop,
        int ret_type
    ]
```

接口处理则是另外一个主要内容，需要在各个节点中调用符号表中的方法，以完善符号表中的各种信息。同时还要进行错误判断，返回报错信息，记录错误数量。

四、总结

1. 实验结果总结

在语法树的基础上完善代码，完成语义分析，我们的编译器能够正确分析本次实验中的全部样例程序。部分代码效率、易读性上仍略有不足，可以继续完善。

2. 分成员总结

吕恒磊：这次实验代码量相较第一次实验有较大提升，主要难点在于设计符号表数据结构和定义与计算各个综合属性与继承属性。我们一起讨论得出了符号表的大致数据结构，根据需求一一定义出所需属性，边实现边改进，最终成功实现了语法分析与类型检查功能。这次实验是一次将理论课知识运用到实践中的极好例子，例如符号表、继承属性、综合属性、自顶向下分析等概念的具体实现等等。摸索实验框架的过程是未知而有趣的，编写代码是高效而有序的，debug 过程也相对没有遇到很大的阻力，感觉收获良多。

康家锐：本次实验的理论支持主要在综合属性与继承属性与对应计算方法的设置。如何实现这些属性并利用其帮助判断对错就是实验的内容。对于语义节点，我们设置了一些属性，并用函数去控制，本次实验中我主要就做了一些表达式相关节点的函数设置，应该说是虽然节点数量多但相对简单的部分。不过即使这样，也确实加深了我对综合属性与继承属性的认识。综合属性要在退出节点时设置，继承节点要在进入时设置就很好的体现了这一点。
这次 debug 过程可以说还挺顺利，这得益于大家一起集思广益效率很高。

王东宇：本次实验中，我们继续完善了上次实验得到的编译器，使其具备了语义分析功能，能够在 CACT 程序上构建语法分析树，并以此为基础实现语义分析。要完成语义分析，需要先构造功能完善的符号表，以保存分析过程中所需的各种信息，保证在遇到错误时能够及时发现并反馈。同时，合作完成代码也让我们体会到了与独自完成代码不同的感觉，收获颇丰。