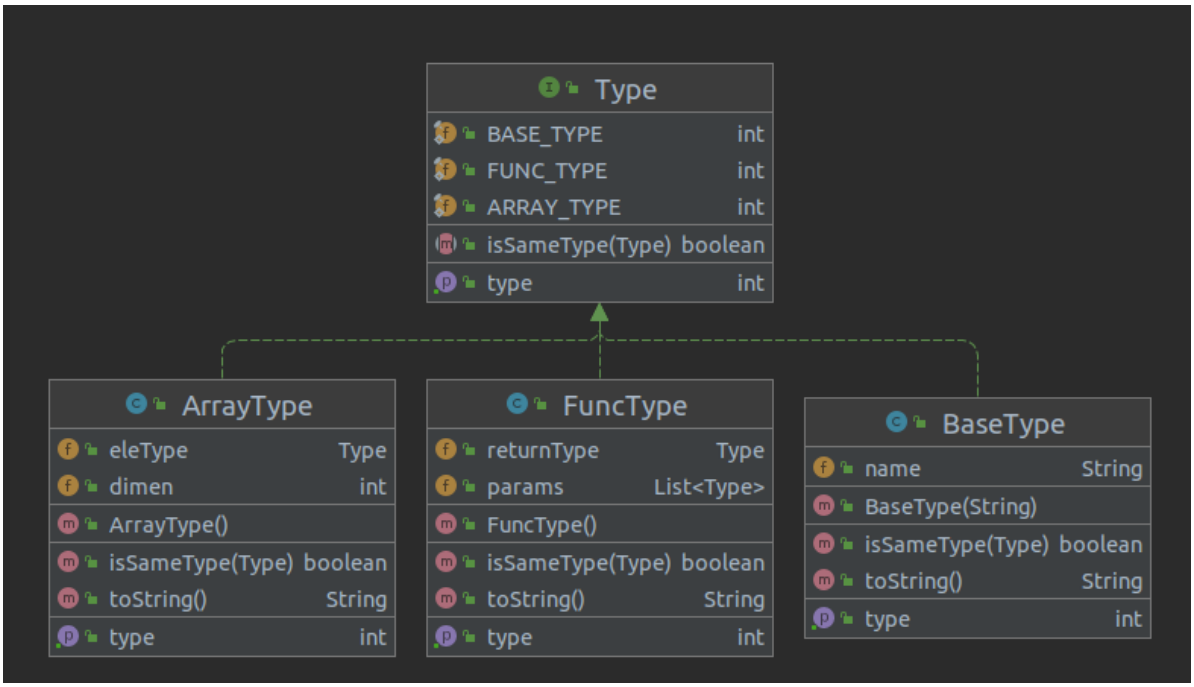
编译原理 Lab3

> 本次实验主要分为两个部分：
>
> 1、对程序进行类型检查，找到所有的语义错误（11种）
>
> 2、若程序没有语义错误，则完成变量重命名后打印语法树

类型检查

设计类型



设计符号表

Symbol类

```
public class Symbol{
    public Type type;
    public String name;
    public Scope scope;
    public List<int[]> usePos;
    public boolean isNeedReplace;
    public int row;
    public int col;

    // 构造函数
    public Symbol(...){}

```

```
    // 增加一个该符号被使用的位置
    public void addUsePos(int row, int col){}

    // 调试使用
    public String toString(){}
}
```

Scope类

```
public class Scope{
    Scope enclosingScope;
    Map<String, Symbol> symbols;
    List<Scope> childScope;

    public Scope(...){}

    // 定义一个符号
    public void define(Symbol symbol){}

    // (全局)解析一个符号
    public Symbol resolve(String name){}

    // (在当前作用域下)解析一个符号
    public Symbol resolveCurrentScope(String name){}

    // 增加一个该作用域的子作用域
    public void addChildScope(Scope s){}
}
```

之后只需要在遍历语法树的过程中，在合适的位置创建作用域，并在合适的位置退出作用域，以及在函数定义、变量声明的对应处理函数中将变量包装成符号加入到作用域的符号列表中。

处理语义错误

需要针对11种错误依次进行分析处理

错误编号	错误说明	分析
1	变量未声明	在使用变量的位置，全局解析该符号，看是否定义该变量，可在`visitTerminal`函数中处理
2	函数未定义	在函数调用中处理，解析被调用的函数名，查看是否被定义
3	变量重复声明	在声明变量的位置：函数形参，常量声明、变量声明三个地方，注意解析时只要在当前作用域下解析（和全局同名变量不冲突）
4	函数重复定义	在函数定义的位置，解析要定义的函数名，是否已经定义过
5	赋值号两侧类型不匹配	出现在有赋值符号的位置：常量声明、变量声明并初始化、赋值语句，处理方式就是取得两边的类型，进行比较判断
6	运算符需求类型与提供类型不匹配	出现在有运算符的地方，注意下标运算符`[]`，处理方式也是获取操作数的类型，检查是否匹配
7	返回值类型不匹配	出现在返回语句中，由于函数类型只有`int`和`void`，如果没有返回值判断是否是`void`，如果有返回值就需要判断`return`后面的表达式返回的类型是否为`int`
8	函数参数不适用	出现在函数调用中，可以先判断形参数量和实参数量，如果相同再进行一个一个的检查判断
9	对非数组使用下标运算符	在出现下标运算符时检查变量是否是数组类型，注意对数组类型变量使用超过其维数的下标运算符个数（例如对一维数组使用两个下标运算符）
10	对变量使用函数调用	出现在函数调用中，判断变量类型是否是函数类型
11	赋值号左侧非变量或数组元素	出现在左值表达式中，如果是赋值语句，判断左侧的符号类型是否为函数类型

> 这一部分最麻烦的是，一个错误会引起连锁错误，但是要求只打印"最本质错误"。
>
> 我在处理的过程中采用：一旦出现错误之后，返回的类型都是`null`，而上层的函数只有在下层函数不返回`null`是才会在这一层进行错误检查，否则直接返回。但是需要注意的是，不是每一个函数返回null都代表出现错误，还需要结合具体情况分析，不过大部分都适用。

重命名

> 为所有与选中变量**生命周期一致**的**同名变量**重命名

思路：

在第一次遍历语法树的过程中，解析变量之后将该变量的位置加入到符号的`usePos`列表中，并且找到需要重命名的那个变量(`isNeedReplace`)。在第二次遍历过程中，将需要重命名的那个符号的`usePos`取出，遍历到对应位置后进行重命名即可。

```
// 获取全部的需要重命名的变量的位置
Queue<Scope> queue = new LinkedList<>();
queue.add(globalScope);
boolean isOver = false;
while (!queue.isEmpty()){
    int size = queue.size();
    for (int i = 0; i < size; i++){
        Scope scope = queue.poll();
        for (Map.Entry<String, Symbol> entry : scope.symbols.entrySet()){
            Symbol symbol = entry.getValue();
            if (symbol.isNeedReplace){
                pos.add(new int[]{symbol.row, symbol.col});
                pos.addAll(symbol.usePos);
                isOver = true;
                break;
            }
        }
        if (isOver) break;
        queue.addAll(scope.childScope);
    }
    if (isOver) break;
}
```

碰到的问题

感觉比较困难的地方主要还是在第一部分语义错误检查，因为出错了之后需要自己想用例去排查，而且“最本质错误”的要求导致bug比较难以找到。