朱家辰 211250044

程序介绍

程序功能

1）识别输入程序存在的语义错误
2）支持函数声明以及相关错误检测
3）支持变量定义受作用域影响
4）检测变量类型采取结构等价

程序编译

在linux环境下，在Lab/code文件夹打开终端，
输入make命令可以生成可执行文件parser，
输入make test命令可以将test1.cmm作为输入程序开始进行分析，
输入./parser $输入程序文件$命令可以指定输入程序进行分析

实现方法

符号表

符号表的实现思路上采取了哈希表的形式，根据符号的名字确定对应的表项
符号表中存储的内容分为3类：变量、函数、结构体。
其中存储了该符号在代码中与本阶段实验相关的信息：名字、行号、类型、变量的初始化信息或函数的参数信息等等
为了实现变量定义的作用域，额外定义了一个栈，记录代码执行到当前阶段的符号作用层级

类型系统

参考讲义中的类型系统给出的参考代码，对类型系统进行了定义，并根据讲义中对类型等价的要求递归定义了类型等价条件

语法树

语法树与上阶段实验不同的核心在于data这个结构体域
为了实现简单与效率提升，这个结构体域同时承担了综合属性与继承属性的职责
具体来讲，每个语法规则需要的属性各不相同，经过研究只需要传递以下几种信息：
1）一个符号对应的指针：例如对于已经定义的某个函数符号是否存在矛盾
2）一个类型对应的指针：例如对于某个即将定义的变量确定其Specifier的类型

3）一个整数：用于计数，例如函数调用传入参数的数量
4）一个字符串：例如某个变量的名字
以上几种全部可以囊括在data这个复合的结构体域中，并且经过我的设计，每个节点只需传递其中之一的信息给其它节点（一定是父亲或孩子节点），即可完成本阶段要求的任务
因此所有节点独立地根据规则使用data这个结构体域，在不同规则中data含义各不相同
这降低了代码可读性，但提高了代码书写和执行效率，毕竟59条规则的语义动作设计已经够麻烦了，再逐一独立设计提高可读性略显强人所难

另外，对于一个新的符号或类型的产生，不外乎以下几种可能：
1）变量定义：对于基础类型，仅有int和float两种，直接加入符号表即可，对于自定义类型以下会提及
2）类型定义：类型中包含了一系列变量定义，只需要新增一层作用域，将变量定义的语句按照原本形式加入符号表，最后将这一层作用域的变量全部弹出，弹出的定义即为这个类型的内容
3）函数定义：同理，函数的形参列表和结构体类型定义事实上是类似的，只需要将新作用域的内容全部加入新函数的参数列表即可，值得注意的是参数列表的变量定义不能立即弹出，需要等待函数体定义结束后才能弹出

最后需要考虑的就是表达式的类型计算，根据表达式的定义逐一判断即可