语义分析

上海交通大學

一、 目标

在之前的作业中，我们实现了词法分析、语法分析，并且在语法分析结束后构建了一棵 AST（Absract Syntax Tree，抽象语法树）。本次作业的目标为利用语法分析器构建的 AST，编写一个语义分析器，实现对 Seal 语言的静态语义分析。

二、 主要任务

1. 遍历 AST，构建符号表，使用符号表来记录源程序中各种名称及其特性信息
2. 静态语义错误检查，错误检查可分为两大类，**符号作用域相关检查和类型检查**

三、 文件说明

- seal-decl.h、seal-decl.cc、seal-stmt.h、seal-stmt.cc、seal-expr.h、seal-expr.cc

本文件存放定义好的 AST 节点，你可以添加额外的说明语句，但不能修改现存的声明。

- semant.cc

这是语义分析的主要文件，也是此次作业**主要需要修改的文件**。它包含了一些预先定义的符号，你可以选择使用或者忽略他们。里面也预置了一些没有实现的函数声明，你**可以选择**觉得其中必要的进行实现，也可以全部实现。

本语义分析器是通过调用 Program 的 semant()方法来运行。Program 的类声明在 seal-stmt.h 中。你所添加到 seal-stmt.h、seal-expr.h、seal-decl.h 中的任何说明语句都会在本文件中被执行。

- semant.h

这是 semant.cc 的头文件，你可以在此处添加你所需要的额外的说明语句

你可能需要花一点时间来阅读相关的代码！

四、 树遍历

在课程作业二的语法分析部分，你的分析器构建了 AST。定义在 AST 节点上的 dump_with_ types 函数，能够以递归的方式打印出整个 AST 的内容。

在进行树遍历时，你需要：

1. 遍历数据结构
2. 管理从树上收集的各种信息
3. 使用这些信息来增强语义

如果你需要添加自定义的信息到 AST 节点，你可以直接编辑 seal-tree.h。

五、 主要工作

语义检查的主要工作在 **README.md 的末尾**有详尽的叙述，这里只在大体上有一个陈述。

- 变量作用域检查

语义检查最主要的部分是命名的管理。你需要确保说明语句对于每一个标识符都是

有用的，并需要跟踪命名所引用的声明语句

符号表可以方便的管理名称和作用域。你可以在项目中使用我们的符号表（查询附属代码包说明文档，了解其功能和使用方法）或者实现自己的符号表。

除了在每个类中固定标识符本身之外，有三种方法可以在 Seal++中引入对象名：

1. 函数的形式参数
2. 函数声明和定义的名称
3. 变量声明语句

对应的，函数的形式参数作用于仅仅位于当前函数声明的语句块中，声明后的函数可以在其他函数中调用，声明的变量名作用域仅仅位于当前语句块中，特别的，可以在所有函数外声明全局变量，这些变量可以在任何一个函数内使用。

除了对象名外，还有函数名。使用任何没有相匹配的声明语句的名称都是错误的（语义分析器不应在发生该错误后中止编译）。

- 类型检查

语义分析器需要能检查是否在需要的地方声明了有效的类型。例如，函数体返回的类型与声明的返回类型必须一致，条件语句的条件位置必须为 **Bool** 类型等等。语义分析器也需要能根据类型规则来确认每一个表达式都有有效的类型。表达式类型规则的细节在 Seal++的参考手册中。

如果一个表达式没有有效的类型，首先语义分析器需要打印一条错误信息，其中包含行号和错误的说明；其次，语义分析器应该继续分析。

此外，请注意检查 main 函数必须存在，且返回值为 **Void**，参数为空。

对于特别的 printf 函数，需要检查是否满足具有至少一个参数，且第一个参数是 String 类型；不能够额外声明别的名为 printf 的函数或变量。printf 字符串的占位符类型检查不要求。

另外，对于每一个函数，我们需要检查，其在最外层部分（函数体的右大括号闭合之前）一定有一个 return 语句。

- 控制流相关性检查

break 和 **continue** 语句，必须是位于某一个循环语句内。

六、 期望的输出

首先可以利用命令 make semant 来构建语义分析器，make clean 可以清理临时文件，每次构建之前，请清理临时文件，保证最新的代码能够生效。

对于错误的程序部分，语义分析的输出是错误信息，错误信息报告方式如下：

semant_error(<tree 节点>) << "出错内容";

这个例程输入一个出错的 AST 节点，返回一个用来写入错误信息的输出流，这个 tree 节点是 AST 树的节点，用于获取其原文的行号。

对于正确的程序，其输出是带类型注释的 AST（记得在语法分析中，有很多的 no_type）。即语法分析中，将所有的类型字段全部填充。而对于不正确的程序，将会报错提示。我们将根据你的语义分析器是否能正确地用类型注释 AST 来评分。

对于错误的程序，要对出错位置和错误进行提示，请查看 README.md 文件末尾，这里有标准语义分析器的输出结果。

对于有些不清楚的情况，请利用提供的 semant 样例文件（位于 semant/semant_example 中）进行语义分析，参考输出的结果判断，如果还有问题，欢迎同学们随时在课程群里提问，需要的话，也会组织线下答疑统一解决同学们的问题。

七、 特别说明

1. 这次的作业不像前两次那样有一些既定的规则，你可能需要灵活地定义一些你需要的数据结构，来实现相关的功能。

2. 另外，对于每一个 Statement 类中定义的 check(Symbol)函数，你可以使用任意的想法来使用这个 Symbol 参数；或者，你类相关成员函数的声明和实现，但是请务必保证——其他人根据你提交的代码可以 make 一个可以工作的语义分析器。

3. 此次实验所采用的代码，源于 Stanford cs143 的开源源代码，大家有兴趣可以在 github 等开源网站搜索 Stanford cs143 分享的作业代码，将对于理解本次作业的目的有很大的帮助。

八、 文件提交要求

要求将原作业目录下的所有文件，放置在一个名为<学号>的目录下，并且将整个目录打包为<学号>_<姓名>.tar 格式。

九、 提交截止时间

请同学们在 2022.1.9 晚上 23:59 之前，将结果提交到 ftp（将会查看文件创建时间，2022.1.9 晚上 23:59 之后的均视为迟交，结果按 0 分处理）。