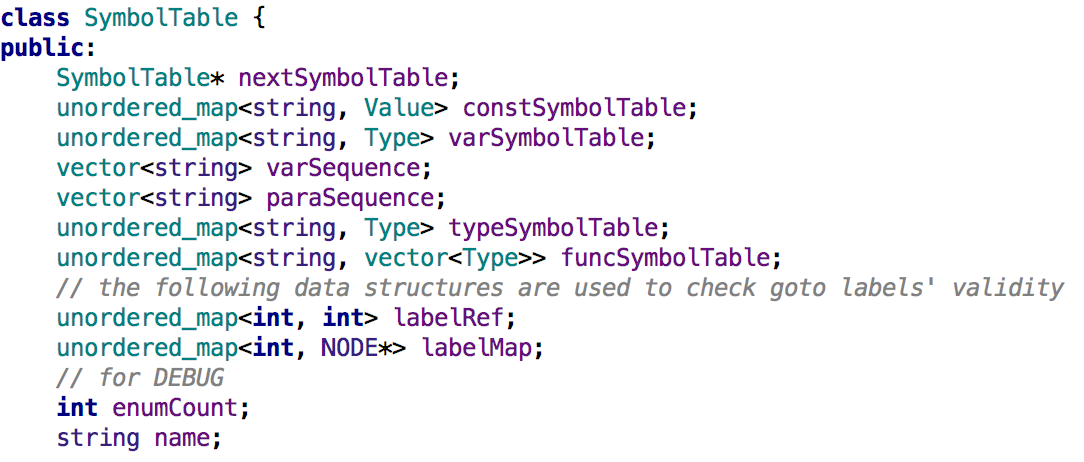
##### 符号表 设计报告

符号表是非常重要的数据结构，他贯穿语义分析阶段的始终，并且会影响到中间代码生成阶段。我们定义了SymbolTable作为符号表的类，其中包含若干个unordered\_map（hashmap），给出符号名即可在常数时间找到对应的符号，相比于手动实现的hash表更加方便快捷。

符号表的主要文件为SymbolTable.h与SymbolTable.cpp，还有一部分类型定义在common.h中，符号表本身定义如下：



其中包含了若干子表，具体信息将在模块设计报告中进行描述。

在语义分析阶段，每一个作用域对应一张符号表，使用链表进行连接，链表头部指示出当前作用域的符号表，在进入/退出作用域是，链表中会添加/删除符号表（但并不进行销毁）。为了避免后续的中间代码生成程序找不到符号表，我们会在需要的AST节点处插入一个指针，指向该节点所需要的符号表。中间代码生成程序即可通过AST节点访问到对应符号表。

##### 语义分析 设计报告

语义分析的程序代码包含在semanticAnalysis.h与semanticAnalysis.cpp中，入口为void semanticAnalysis(NODE\* ROOT)，该函数主要接受一个来自语法分析程序所生成的AST根节点。该节点包含了程序源文件的所有信息。然后在AST上进行遍历，检查是否存在语义错误，如果存在，通过LOGERR函数输出错误，最后利用semanticAnalysisError这个全局变量，指示是否在语义分析阶段发生了错误，该变量非0，表示存在语义错误，中间代码生成可不用执行，直接退出。

语义分析阶段的另一工作是进行类型的检查与自动提升，为后续的中间代码生成阶段提供必需的信息，语义分析程序会计算表达式的类型，并将其附在AST节点上。虽然最后semanticAnalysis函数无返回值，但在调用完此函数后，在确保可以进行后续的代码生成之外，AST本身将会增添额外信息。

语义分析的大致流程图如下：



详细过程将在模块设计报告中进行描述。