lab2_实验报告

21311111 乔羿童 qiaoyt3@mail2.sysu.edu.cn

编译和测试

1. cd Code/：进入源代码和 Makefile 文件所在的路径下
2. make：使用 make 命令对程序代码进行编译构建
3. ./parser ../Test/xxx.cmm：对指定的 C--代码 xxx.cmm 进行分析，注意测试文件的路径要正确。也可以修改 Makefile 文件中的 test 部分的内容，然后使用 make test 命令进行测试。第二种方式可以同时输出多个测试文件的测试结果。

1. 符号表

支持**多层作用域**的符号表：采用**十字链表 + Open Hashing 散列表**。

前者可以表示作用域的层次关系，后者可以解决哈希冲突。参考项目书中的说明并结合代码，对该符号表的维护风格是 **Imperative Style** 的，即始终在单个符号表上进行动态维护。 符号表是用来存储源代码中所有符号信息的数据结构，包括变量、结构体、函数等。Hash 函数：PJW，即代码中的 `hash_pjw()`，一个常见的字符串散列函数。

1.1 符号表的操作

填表操作

- 当向符号表中插入符号（符号表条目类型 `Entry`）时，调用 `insertSymbol()`。
- 计算符号的哈希值，先插入对应槽位的下挂的链表的表头，然后再插入对应层次的链表的表头。

查表操作

在填表前需要查表，检查在某一作用域内是否存在名字被重复定义的情况。查表的时候如果定位到某个槽位，则按序遍历该槽下挂的链表并返回槽中第一个满足条件的变量。

- 查找所有符号： `findSymbolAll()`在符号表中查找指定的符号，无关作用域。
- 查找同一层次的符号： `findSymbolLayer()`只在当前作用域内（`symbol = symbol->layerNext`）查找指定的符号。
- 查找函数符号： `findSymbolFunc()`专门用于查找函数类型的符号。

1.2 作用域的管理

- `Entry`： 是符号表中的一个条目，表示一个符号。除了符号的基本信息 `name` 和 `type`，还包含两个指针：`hashNext` 指向同一槽位的下一个条目，解决散列冲突，`layerNext` 指向同一层次的下一个条目，链接同一作用域内的符号。
- 当新进入一个语句块时，调用 `pushLayer()`，为这该层语句块新建一个链表，串联该层次中新定义的全部变量。
- 当离开一个语句块时，调用 `popLayer()`，通过链表的指针操作，删除对应层次，并结合 `while` 循环不断调用删除符号函数 `delSymbol()`，顺着代表该层语句块的链表指针 `layerNext` 将该层次的符号全部删除。

2. 变量类型

在 `semantic.h` 中为语义分析阶段存储和管理类型信息所需的数据结构进行定义。

- 对于变量类型，首先使用**结构体**进行类型构造，构造出**结构体域链表节点**（包含域的名字和类型，指向下一个域的指针），**结构体类型**（包含结构体名字和指向结构体中的第一个字段的指针。即结构体的实现是通过多个**结构体域链表节点**组成的链表），**函数类型**（包含函数名称，函数返回值类型，参数个数，指向参数链表头节点的指针，函数是否已经定义的标识以及所在行数）以及符**号表条目类型**。

- 避免直接使用指针的复杂性：采用一系列的 `typedef` 最终指向上述构造类型的指针，比如 `Function` 为函数类型结构体原型 `Function_`定义了别名，然后其又定义为 `Function_*`，即指向 `Function_`的**指针**的别名，方便后续代码中的引用，不需要频繁地使用指针语法。

- 用 `union` 来存储不同种类的类型信息，例如基本类型、数组、结构体和函数类型。其中数组类型新定义为结构体 `array`，其中包括元素类型和数组大小。

3. 结构等价

实现了结构等价判断方法，包括基本类型（`ENUM_BASIC`）、数组（`ENUM_ARRAY`）、结构体（`ENUM_STRUCT`）和函数（`ENUM_FUNC`）。相对复杂的，对于结构体，当两个结构体中所有对应的字段类型都等价，且字段数量完全相同，结构体类型被认为是等价的。而对于函数类型等价，要保证参数数量、参数类型和返回类型完全相同。

4. 语义分析实现简述

在实验一构建的 AST 的基础上，对 AST 进行遍历以进行符号表相关的操作。实验代码选择将**语义分析**相关的代码放到单独的文件 `semantic.c` 中。由于实验项目书中假设输入文件中不包含任何的词法或语法错误（除特殊要求），因此在 `main.c` 中选择在没有出现词法和语法错误的时候调用函数 `semantic_analyse(root)`进行语义分析。

该函数首先调用 `initSymbolTable()`初始化符号表，然后调用 `Program(root)`即调用 `ExtDefList()`开始递归地处理定义列表，处理过程中进行对符号表的操作。等到函数递归调用结束之后，根据构造完成的符号表，调用 `check()`函数，通过对符号表中的每个槽下面的链表进行按序遍历，检查符号表中的条目以确定是否存在**函数被声明但是没有被定义**的情况。

4.1 递归遍历 AST

代码通过**递归遍历 AST** 的方法，代码检查 AST 中的每个结点，并且在必要的时候更新符号表，同时对可能出现的错误（如**类型不匹配，重复定义、未声明的使用**等）进行打印。

递归处理：以对定义列表的处理方法 `ExtDefList()`为例，在函数中先调用外部定义处理方法 `ExtDef()`，后面递归调用定义列表的处理方法 `ExtDefList()`。参数列表处理方法 `VarList()`、扩展定义列表处理方法 `ExtDefList()`同理。

通过开始向 `ExtDefList()`传递 `root` 根节点，**自顶向下**遍历整棵语法树。实验一中生成的 AST 具有层次结构，每当遇到非叶子节点时，根据其子节点 `children` 的信息来判断其内容和进行的操作，结合符号表中已有的信息进行错误判断。 相当于为 AST 的每一个非叶子节点定义一个对应的函数，对以此节点为根的子树进行分析。

4.2 错误类型检测

以一段 `int main() {}` 的代码片段生成的 AST 为例：

```
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE: int
      FunDec (1)
        ID: main
        LP
        RP
      CompSt (1)
        LC
        RC
```

在处理单个定义 `ExtDef(Node *root)`时，`main` 函数 `root->children[0]->name` 为 `Specifier`，`root->children[1]->name==FunDec`，`root->children[2]->name` 为 `CompSt`。使用 `strcmp` 方法进行比较，`root->children[1]->name==FunDec`，说明出现函数的定义或声明。 则首先使用定义的 `FunDec` 函数提取出函数名和参数信息。调用 `findSymbolFunc` 在符号表中搜索该函数名，如果不为空说明已经存在，则检查 `hasDefined` 属性确定其是否已经有定义，结合 `FunDec` 后方跟着是函数体（`Compst`）还是声明语句分号（`SEMI`），来确定**函数重复定义、声明和定义冲突、声明和声明冲突等情况**并进行处理。如果该函数名未空则说明是首次出现的函数声明或定义，则需要相应地进行符号表的插入操作，如果是函数定义则还需要对函数体进行递归处理，包括检查变量声明和处理所有语句。

- 对于**函数返回类型**的判断，在单独语句的处理 `Stmt` 中进行，首先判断 `children[0]` 是否为 `RETURN`，符合条件则进行类型的比较。 函数定义的返回类型在符号表中已经存储，而实际返回的类型则要对 `return` 的 `children[1]`进行 `Exp` 分析以得到。
- 遇到**结构体**描述符 `StructSpecifier`，则根据其子节点 `STRUCT`，`OptTag`，`LC`，`DefList`，`RC` 进行判断和处理，判断**直接使用未定义的结构体**等错误。
- 遇到**表达式 Exp** 节点，说明该结点及其子结点们会对变量或者函数进行使用，同样根据 `children` 查符号表以确认这些变量或者函数是否存在以及它们的类型是什么。包括**错误地对非结构体变量使用'.'操作符，访问结构体中未定义过的域**。
- 对于**数组错误**的检测，在 `Exp` 的子节点中，`children[1]`即为`'['`，判断 `children[0]`的类型来确定**数组访问操作符使用对象**是否正确，判断 `children[2]`的类型来确定**数组访问操作符内是否存在非整数**。
- 对于**赋值操作**的判断和**二元运算操作**的判断，同理通过 `children` 确定赋值号左右的符号的信息，来进行不匹配的检测。
- 对于**使用未定义的变量和函数**的检测，通过当在 `Exp` 中检测到 `ID` 时，根据 `root->childNum` 确定 `ID` 是变量还是函数名。子节点数量为 `1` 则是变量，调用 `findSymbolAll` 从符号表中查找 `ID` 是否存在。否则为函数名，调用 `findSymbolFunc` 进行查找，再调用 `findSymbolAll` 进行查找，可以判断**`()`的使用对象是否正确，使用的函数是否定义**。后续根据 `childNum==4` 判断函数是否有参数，进行比较以判断**函数调用时实参与形参是否匹配**。
- **重复定义**：对于简单变量 `VarDec`，直接查找其名称是否已在当前作用域或全局作用域中定义进行判断。
- 非叶子节点 `Def`，通过其 `children` 节点 `Specifier`，`DecList`，`SEMI` 进行分析。

总之，语法分析流程的关键是 **AST 的遍历**，对节点的信息进行判断和处理。