语义分析

梁宇钦 151180076

May 27, 2018

1. 功能实现

1.1 功能实现情况

实验实现了基本要求的假设 1, 2, 6, 7 以及选择完成部分的要求 2.1, 2.2, 2.3。对于给出的样例均能正确报出相应的错误。

1.2 功能实现方法

对于函数声明的允许，修改（增加）了一个产生式：

ExtDef : Specifier FunDec SEMI

对于其他功能的实现，主要依赖与符号表。

1.2.1 符号表数据结构

符号表以哈希表作为基础，是一个多层表结构 (即子表结构)。通过多层的符号表结构，可以很容易实现变量作用域的要求。

对于任意一个程序代码文件，构造的符号表可以用下图表示。

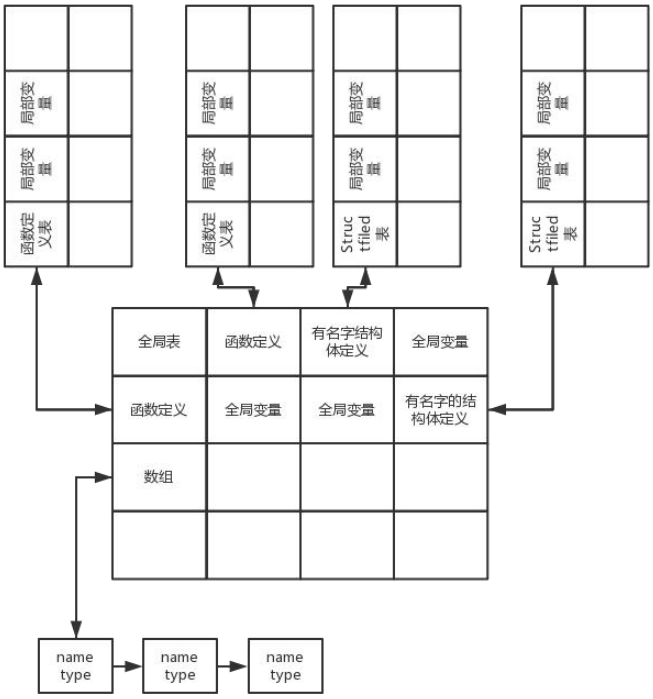

图 1-1 符号表示意图

1.2.2 关于 hash

本实验采用的 hash 函数以及 hash 表的基本表是第三方的 uthash。文件放在 3rdparty 目录下面。

uthash 是一个宏定义的 C 语言 hash 表。它是一个动态空间的 hash 表。即 hash 表的空间不是固定，当表里面的元素达到设定的阈值，表的空间会扩大为原来的两倍，并重新计算 hash。

因此，本实验采取的多层 hash 的数据结构并不会占用很多的内存，主要还是取决与符号元素的数量。

2. 编译和测试

这部分内容在文件“README.MD”中有详细说明, 这里仅做简要说明。

2.1 目录树

```
Semantic.d
├── 3rdparty
│   └── uthash.h
├── lexical.l
├── main.c
├── Makefile
├── README.md
├── runtests.sh
├── sample.d
├── semantic.c
├── semantic.h
├── SymbolTab.c
├── SymbolTab.h
├── syntaxtree.c
├── syntaxtree.h
├── syntax.y
```

本次实验的所有内容都在 Semantic.d 目录下面。主要文件包括 lexical.l, syntax.y, syntaxtree.h, syntaxtree.c , SymbolTab.c, SymbolTab.h, semantic.h, semantic.c 和 main.c 等文件。其中 lexical.l 是词法文件, 用 flex 编译；Syntax.y 是文法文件, 用 bison 编译；syntaxtree.h 和 syntaxtree.c 是语法树的数据结构文件；SymbolTab.c 和 SymbolTab.h 是符号表数据结构文件；semantic.h 和 semantic.c 是语义分析文件； sample.d 下面是样例。

2.2 编译

直接在终端输入命令:

```
make
```

就可以了编译了。

2.3 测试

2.3.1 测试已经提供的样例

要测试实验指导提供的样例, 也是非常简单的. 直接运行脚本 “runtests.sh” 即可. 建议采用以下的命令运行脚本:

```
sh runtests.sh
```

其结果会会打印屏幕上。

2.3.1 测试其他未提供的样例

命令格式如下:

```
./parser filename.c
```

其结果会打印屏幕上面。