编译原理报告——实验2

刘志刚
匡亚明学院
141242022
njuallen@foxmail.com

实验进度

- [x] 基本要求
- [x] 要求2.1：函数声明
- [x] 要求2.2：嵌套作用域
- [ ] 要求2.3：结构等价

编译运行

进入Code目录下，输入`make`即可编译，生成的可执行文件为`parser`。
执行方式：`./parser test.cmm`。
输入`make test`即可运行Test目录下的所有测试。

实现说明

函数声明

为了支持函数声明，我添加了一条产生式：

```
ExtDef : Specifier ExtDecList SEMI {
}
```

```
  | Specifier SEMI {
  }
  | Specifier FunDec CompSt {
   // function definition
  }
  | Specifier FunDec SEMI {
   // function declaration
  }
```

同时，对于函数，我使用了两张符号表。分别存放函数定义及声明。每次有新的定义或声明出现，就把两张表分别查一下，看有没有冲突。另外在处理完所有语法树节点后，我们要比对定义表及声明表，看是否有声明了但是没定义的函数。

嵌套作用域

为了支持嵌套作用域，我使用了讲义上推荐的十字链表和栈的那种结构。作用域的嵌套可能出现在结构体、函数的定义中。因此每当遇到结构体或函数的定义，我们就新建一个作用域。另外值得注意的是，我们在处理函数时，由于形参也有名字，形参也要放到这个新的作用域中。如果接下来有函数体，即这是函数定义，则这个作用域要在处理完函数体后删除。如果没有函数体，直接就是分号，则这是函数声明，作用域可以直接删除。

符号表

我使用了四张符号表，分别是变量名表，结构体名表，函数定义表，函数声明表。

匿名结构体

我们的文法是支持匿名结构体的，匿名结构体的出现为我们的类型比较带来很大的麻烦，例如：

```
struct {
    int c;
}a;

struct {
    int c;
}b;
```

这里的a和b类型是不同的。为了方便处理，对于匿名结构体，我给他们赋上全局唯一的名字，形如`<anonymous>_id>`，其中id是一个单调递增的数字。匿名结构体有名字之后，我们就可以把它插入到结构体符号表中，当作普通结构体进行处理。

内存管理的问题

在这次实验中，我们要实现一个并不算复杂的数据结构，要管理大量的指针。我发现，如果在设计阶段，就要想搞清楚这些数据结构的生存期，什么时候该使用，什么时候该释放，使用会不会出问题，将会是个巨大的思维负担，给设计带来了枷锁。所以我采取了懒惰的手段，只malloc，不free，避免出现内存问题。事实证明还是自动内存管理对程序员更友好啊！