L3 语义分析 实验报告

181250015 陈彦泽

1 实现功能

利用L2生成的语法分析树和属性文法，完成错误类型1-17的检测

2 实现方法

1. 使用L2生成语法分析树

   本人L2的main函数写在`syntax.y`中，L3中为了实现方便，提取到了`main`文件中，因为对C比较熟悉，就用改成了`main.c`

2. 定义符号表`hashtable.c`，实现基本的查找、插入功能

3. 在`semantic.c`中进行属性文法的语义分析，基本思想就是DFS遍历生成的语法树，将**继承属性**以参数的形式传给**下层节点的解析函数**，如果有**综合属性**，就使**下层节点的解析函数**的**返回给上层节点的解析函数**

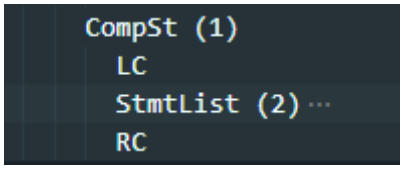
3. 一些实现细节

3.1 L2的遗传问题

在L2中的`addChild`方法中，规则是如果节点是NULL就不插入为子节点

```
void addChild(Node* parent,Node* son){
      if(parent == NULL||son==NULL){
        return;
     }
}
```

这可能导致少添加节点，比如`CompSt : LC DefList StmtList RC`



这种情况下因为`DefList==NULL`不会被添加，但实际上应该要添加的，不然L3在分析的时候，由于`n.child[1]`有两种情况(DefList、StmtList)，无法通过`n->child[1]->identifier`进行判断，会增加很多不必要的麻烦。所以去掉`son==NULL`的判断，并响应的修改了L2中递归打印语法树的方法。

3.2 各种NULL处理

```
cyz@cyz-ubuntu:/mnt/hgfs/share_ubuntu/compilers/Lab$ ./parser tests/lab3/my.cmm
Segmentation fault (core dumped)
```

在L3实验中，最常见的错误就是Segmentation fault了。在这次实验中，基本上遇到的所有情况都是因为递归调用的函数返回NULL，但上层函数没有对NULL进行了判断，对其进行了`->`操作导致的。

此外，本实验中基本所有地方在`printError`后就会`return NULL`，但为了**错误恢复**也有例外，比如在`VarDec`中，这是为了在定义函数时候，如果参数的命名重复了，会报错，但是函数的参数关心的类型信息依然会被保留，这使得下文在函数调用时候可以进行参数的检查。

3.3 为什么需要FieldList?

本次实验不需要实现作用域，一开始的想法就是直接遇到定义就查表/添加，但是后来发现一些需求必须使用一个数据结构将域存起来：

- 函数的参数——为了在函数调用时候进行参数检查
- 结构体的域——为了比较两个结构体是否结构等价

3.4 退出时机

在本次实验中，发现类型检查的过程中，遇到错误退出时机也需要考虑，以讲义中要求说明的例2为例

> 在遇到函数定义时，不会检查第二个函数定义内发生的错误

也就是说在FunDec中首先要检查ID，如果有错误就直接返回NULL

```
if(check(n->child[0]->value)==1){// 先检查是否重复了，重复就不加入
    printError(ERROR_FUN_REDEF,n->child[0]->line,n->child[0]->value);
    return NULL;
}
```

而在ExtDef中，对FunDec的返回值进行检查，如果是NULL，就就算后续有ComSt，也不进行检查了

```
Function func = FunDec(tmp,t);
if(func==NULL){ // 有错误，比如重复定义了，直接退出
    return;
}else{
    ...CompSt(n->child[2],t);
}
```