AST 设计文档

AST 的构造主要可以分成以下几个步骤：

- 修改 P4,包括.lex 和.y,在.y 中加上语法制导的翻译，使之成为 AST 的文法。
- 修改 common.h，增加 AST 的节点和函数，使之能支持 C0 语言。
- 针对 common.h 声明的函数，在 ast.c 中写出对应的实现。

  此外，还有一些细节上的修改，比如增加 op.h 中的运算符，使之能完全支持 c0 语言。

1 AST 文法

AST 的文法在 P4 的基础上加上了语法制导的翻译，主要是在文法符号下增加了该节点的创建函数，P4 的文法基本上没有改变,只有一个细节的修改把

```
CDECL       :CONSTSYM INTSYM IDENT ASGN NUMBER CDELF ';'
            ;
```

改成了

```
CDECL       :CONSTSYM INTSYM ASSN CDELF ';'
            ;
ASSN        : IDENT ASGN NUMBER
            ;
```

以方便写语法制导的翻译。

具体的文法在 C0AST.Y 中。

2 AST 结构 <common.h>

在 common.h 中，增加了

KVdecl,

KVdelf,

KCdecl,

KAssn,

KCdelf,

KFunctionDef,

KMainDef,

KCompStat,

KStatf,

KStatif,

KWlop,

KFunctioncall,

KRelation,

这些不同的类型，他们分别对应到 c0ast.y 中的文法符号，此外还增加了一些结构体以支持这些类型。common.h 的最后一部分给出了针对不同类型节点的创建和删除的声明，以及 AST 的创建和删除函数。

3 AST 的实现代码 <AST.C>

Ast.c 中针对每种类型分别给出了创建和和删除函数，每种类型的函数的 实现都差不多。还有 setloc 函数，用来记录节点位置，此外还有一个比较重要的 dump 函数，它用输出相应的源码以检测 AST 的对错。它也是给每种节点类型给出一个相应的输出。在 ast.c 中我定义了一个变量 i，用来记录每个语句的缩进是多少，使程序看起来比较直观。

c0.lex 和 op.h 也做了一些小小的改动，c0ast.lex 中主要是将值赋与相应的节点，op.h 进行了扩充，加入了一些常用的关系运算符。

4 实验心得

这一次实验是目前写过的实验中量比较大的一次，而且 common.h 和 ast.c 里面要写的东西虽然 挺多，但几乎是重复的工作，刚开始做就花了一下午的时间敲键盘。写完以后编译错误很多，大体上都是敲错字的缘故，比如把 functioncall 写成 fuctioncall 等。这次实验有一大部分的时间都在调这种错误了。

实验中还发生了一件很奇怪的事情，我编译时，编译器里面报 destroylist 和 listclear 这两个函数 util.h 里面和 list.c 里面有类型冲突，我改了很久也没改对，后来我把自己写的 p5 里面的东西 copy 到老师的 bison_example 里面，把里面我需要用的函数全部用自己写的替换掉，把 makefile 也替换掉，然后那个错误就没有了。我把 bison-example 改名叫了 p5,这时候这个 p5 和原来的 p5 是一模一样的，可是用原来那个还是报错，用一模一样的新的 p5 就不报，我完全没弄明白是怎么回事。

比较幸运的是编译通过后结果也是对的，不需要再纠结的修改文法。我很感谢自己在一开始写 c0 文法的时候比较用心，所以 P4 和 P5 文法几乎都不需要修改，省去了很多调试的痛苦。