编译原理实验一实验报告

171860607
白晋斌
810594956@qq.com

任务号:14

目录

实验要求

实验一要求通过标准输出打印程序的运行结果.对于那些包含词法或者语法错误的输入文件,只要输出相关的词法或语法有误的信息即可. 程序在输出错误提示信息时,需要输出具体的错误类型、出错的位置(源程序行号)以及相关的说明文字(不做具体要求).

程序要能够查出 C--源代码种可能包含的下述几类错误:

1.词法错误(错误类型 A):即出现 C--词法中未定义的字符以及任何不符合 C--词法单元定义的字符.

1.2:识别指数形式的浮点数.若输入文件中包含符合词法定义的指数形式的浮点数(如 1.05e-4),程序需要得出相应的词法单元;若输入文件中包含不符合词法定义的指数形式的浮点数(如 1.05e),程序需要给出输入文件有词法错误(即错误类型 A)的提示信息.

2.语法错误(错误类型 B).

对于那些没有任何词法或语法错误的输入文件,程序需要将构造好的词法树按照先序遍历的方式打印每一个结点的信息.

测试环境

Ubuntu 16.04
gcc version 4.7.4 (Ubuntu/Linaro 4.7.4-3ubuntu12)
flex 2.6.0
bison (GNU Bison) 3.0.4

运行方法

make clean
make
make test (或者./parser xxx.cmm,cmm 需要在 parser 同一目录下)

实验步骤

1.词法分析

1.1.按照附录 A 中的 7.1.1 实现正则表达式,即 flex 中的 definitions 部分,因选做要求,这里对八进制数和十六进制数、注释不做相关词法识别,故遇到这两种情况程序会报词法或语法有误.这里遇到了 BUG1,错误与解决方案详见最后一部分.

1.2.rules 部分重要的是顺序,特殊的放前面,一般的放后面,参见 BUG2.

1.3 每个词法相当于一个结点,user subroutines 相当于将读到的每个词封装成一个结点,这个本来是 rules 部分的内容,因重复性较高,这里提取出来单独封装成函数.需要注意的是这里结点的数据全部用 int 来保存,之后分析可以将 int 作为索引查表得到具体内容,这样的好处是提高存储和检索效率,事实上,可以进一步优化为按位存储(计算机系统课上,i386 很多东西都是按位标识)

2.语法分析

2.1.按照附录 A 中的 7.1 与 7.2 对整个语法进行理解,从而实现 rules 部分.

2.2.**语法树**.在构造多叉树时,考虑到我们要按照先序遍历的方式打印词法树,这里创造性地使用了森林的结构,同一层互为兄弟,展开层的第一个结点为上一层的子女.在处理ε的时候,我们遇到一点小问题,开始直接将父结点设置为 NULL,但是,考虑语法 A->BCD,C->ε,若 C 被设置为空,我们的森林结构将出现问题!无法访问到 D,因此在这里我们又新增了ε的结点,只是在打印时对ε结点做特殊处理,不进行打印.详见 BUG5.特殊地,当 cmm 文件只有空白符时,仅打印 Program 且行数为 EOF 行,这里用 yylineno 即可.

2.3.**错误恢复**.**这是我认为实验一最难的部分**.实验手册对此介绍很少,经过对附录 A 中 7.1 与 7.2 的消化理解,在草稿纸上构造各词法的树状结构图,最终按照以下原则适当添加了一些错误恢复的 error 标记.

值得注意的是,错误恢复并不能 100%保证可以准确识别出每一行的词法或语法错误,开始我以为是自己的 error 不够完善,但经过测试 gcc 发现,对于一些变态级的测试用例,如

```
int main(
{ int a=.e;
intt b=1
};
```

gcc 也仅能识别出第一行的错误,因此这里只是尽可能按照一些简单原则增加了错误恢复.

简单原则包括:

(1)文法的末尾不做错误恢复,因为文法末尾出错的话,同步词必然在上一层,对上一层做错误恢复即可.

(2)对于一个文法 A->B SEMI,若我们通过理解可以推断出 B 必然是写在一行之内的内容,如 EXP 相关表达式,我们不作错误恢复处理,统一由上层的 error SEMI 进行.(这也是依据了一行报一错的原则)

(3)对于 A->BA 的文法,我们对 B 进行错误恢复,否则一旦出错,可能会跳过多个 B.

(4)对于可被分解的文法,我们在遵循原则 3 的基础上尽可能地在其下层进行错误恢复,从而实现提前错误恢复,解决诸如 int a=.e; /n innt b=1; 这样的连续两句报错.

(5)在不产生冲突的前提下,我们尽可能地对终结符进行错误恢复.并对前四例产生的冲突,删除部分错误恢复语句以实现程序的优美性(尽管不太影响程序运行).

遇到的问题

一路遇到很多问题,这里挑选一部分经典问题与对应解决方案列出:

BUG1: 正则表达式中一些符号在用到本身的含义时,需要转义,如* . ? + $ ^ [ ] ( ) { } | \ /

BUG2: Lex - Warning, rule cannot be matched

flex 是从前往后依次匹配,因此把 ID 放前面就会覆盖点关键词,所以应该特殊的放　　前面,一般性的放后面,参考网站:

https://stackoverflow.com/questions/36955210/lex-warning-rule-cannot-be-matched

BUG3: C 语言错误：expected declaration or statement at end of input

可能错误：1.某一个函数或者变量没有在使用之前声明。2.某个地方少了个括号。(这个最坑)

BUG4: 当结构体中有指针时,结构体不能直接用"="方式的默认构造函数来赋值.

BUG5: nothing cannot set as null treenode, or there will be Segmentation fault (core dumped)

BUG6: C 语言中使用 struct 要时刻带上 struct 关键词,参考网站:

https://stackoverflow.com/questions/2743949/useless-class-storage-specifier-in-empty-declaration

BUG7: implicit declaration of function 问题解决:

C 语言程序编译后出现警告： warning: implicit declaration of function ‘xxx’ [-Wimplicit-function-declaration]

原因：相关的头文件没有声明这个函数，在相关头文件中声明即可