Compiler Lab Report

刘嘉欣

October 13, 2023

1 功能与实现

1.1 词法分析

1.1.1 词法单元的识别

可以识别词法单元：C–的 INT，FLOAT，ID 及其他基本词法单元，另外实现了注释的识别和过滤。

1. INT: 0|[1-9][0-9]*
2. FLOAT: [0-9]+.[0-9]+
3. ID: [_a-zA-Z][_a-zA-Z0-9]*
4. 其他符号：对于一些特殊符号，正则表达式可以使用转义符号或带上双引号，如：PLUS->"+", STAR->"*"

注释的过滤：C–的注释有两类，一是使用双斜线 "//" 进行单行注释，在这种情况下，该行在"//" 符号之后的所有字符都将作为注释内容而直接被词法分析程序丢弃掉，对应于正则表达式中的".*"。而对于多行注释的正则表达式，由于在"/*" 与之后最先遇到的"*/" 之间的所有字符都被视作注释内容，包括"*"，故采用上面的正则表达式进行识别。其中，"/*" 对应注释符号的左半边，我们希望被注释的文字其中没有"*/"，故可以分为两种情况：一是可以是不包含"*" 和"/" 的任意字符，二是一个或多个"*" 字符，但是后面紧跟的字符不能是"/"。注释的右边是一个或多个"*" 跟着一个"/"，故注释的表示如下。

```
"/*"(([^\*])|((\*)+[^\*/]))*(\*)+"/" {}
"//".* {}
```

1.1.2 词法错误

1. 未知符号：在规则的最后加上"." 表示对任意字符的识别，找出"Mysterious character" 的错误
2. 错误的 ID：[0-9]+[_a-zA-Z]+[0-9]* 表示对错误的 identifier 的识别
3. 错误的整数类型：由于未实现八进制整数，0[0-9]+ 是错误的整数类型
4. 错误的浮点数的识别：浮点数小数点前后都有数字，且小数点只能有一位。由于未实现浮点指数，其表达式也是错误的

1.2 语法分析

1.2.1 语法树的生成

我们略去大部分的定义细节，讲述语法树这棵多叉树生成的步骤。
首先，定义树结点，结点包含两个指针：fir 和 sib，分别指向第一个孩子和结点的兄弟。

```
typedef struct _node_t{
    struct _node_t *fir, *sib;
} node_t;
```

对于插入操作，若父节点的 fir 为空，则放置于此；否则找到 fir 的" 最后" 一个兄弟，再让它的 sib 指针指向被插入的结点即可。
打印结点信息需要先序遍历，采用递归的方式实现即可，先打印当前结点的信息，再递归的打印 fir 结点和 sib 结点的信息，比较简单。
** 销毁操作暂时还未实现，看后续是否需要。

1.3 语法错误的识别与恢复

书写一些包含 error 的产生式，可以使得分析器能够识别出一些简单的语法错误并进行恢复，举一些例：

1. CompSt->error RC 可以识别一般的语句块错误
2. Stmt->error SEMI 可以识别一般的语句错误
3. Def->Specifier error SEMI 可以识别函数体内变量定义不能缺失 DecList
4. Args->Exp COMMA error 可以识别不合法的实参
5. Exp->Exp LB Exp error RB 可以识别不合法的数组调用
6. ..

2 编译方式

采用 Makefile 进行编译，执行的命令如下：(其中，syntaxTree.c 和 syntaxTree.h 保留了语法树的定义以及相关操作)

1. flex -o lex.yy.c lexical.l 使用 Flex 编译 lexical.l 文件，生成 lex.yy.c
2. bison -o syntax.tab.c -d -v syntax.y 使用 Flex 编译 syntax.y，生成 syntax.tab.c 和 syntax.tab.h 两个文件，其中 lexical.l 引用了 syntax.tab.h
3. gcc -c syntax.tab.c -o syntax.tab.o 编译 syntax.tab.c，生成 syntax.tab.o
4. gcc -std=c99 -c -o syntaxTree.o syntaxTree.c 编译 syntaxTree.c，生成 syntaxTree.o
5. gcc -std=c99 -c -o main.o main.c 编译 main.c，生成 main.o
6. gcc -o ./parser syntax.tab.o syntaxTree.o main.o -lfl -ly 把 main.o 和 syntax.tab.o 和 syntaxTree.o 放到一起进行编译成最终的可执行文件 parser