哈尔滨工业大学

编译系统 2022 春

实验一

学院：	计算学部
姓名：	冯开来
学号：	1190201215
指导教师：	陈鄞

一、 实验目的

a) 巩固对词法分析与语法分析的基本功能和原理的认识
b) 能够应用自动机的知识进行词法与语法分析
c) 理解并处理词法分析与语法分析中的异常和错误

二、 实验内容

(一) 实验环境

Ubuntu 12.04, kernel version 3.2.0-29; GCC version 4.6.3
GNU Flex version 2.5.35; GUN Bison version 2.5

(二) 程序功能

1. 数据结构

本次实验用到的数据结构很简单。一个结构体数组，里面维护了节点的名称（语法单元，词法单元的名称），行号（语法单元用），num（儿子的数量），childList（儿子列表），一个联合体（若为 TYPE，ID，INT，FLOAT，存放对应的数据）。还有一个数组，用于存放这棵树，并设置了 flag 是否为 child（打印的时候有用）。

2. 词法分析

词法分析属于整个实验甚至是语法分析中的一部分，在一开始写的时候先写的词法分析。**单独对于词法分析来说**，我做到了以下功能：

- 能够正确识别十进制数、八进制数、十六进制数并能进行**进制转换**。
- 能够正确识别浮点数、指数形式的浮点数。
- **可以判断出错误的八进制数、十六进制数、浮点数、指数形式数。**
- 能够**识别所有的注释符号，并判断出错误的注释符号。**
- 能够识别所有的词法单元 ID。
- 能够特别的识别出 ID 中的 TYPE（int 和 float）

因为其中正确识别十进制数、八进制数、浮点数等都是基本内容，本实验做的比较特别的内容在于实现了判断出词法分析中的错误类型。这里用到的方法就是对于错误的八进制数、十六进制数再进行定义。比如八进制数首位为 0 但是后面的数中不能出现 9，所以对首位为 0 且后面出现 9 的数定义为 INT8_ERROR。同理，对于 INT16_ERROR 我们认定在 0x 后面出现[g-zG-Z]都为错误。

```
INT8_ERROR 0[0-7]*[8-9]+[0-7]*
INT16_ERROR 0[Xx][a-fA-F0-9]*[g-zG-Z]+[a-fA-F0-9]*
```

对于指数形式的浮点数，略有些复杂，因为形如.5E03 和 10.e3 都是对的，但是对于形如 10.e 和.e 和 10.和.05 都是错误的。定义如下图：

```
FLOAT_ERROR [0]+(0|[1-9][0-9]*)\.[0-9]+|\.[0-9]+|[0-9]+\.
SCIENCE_ERROR [0-9]+[Ee][+-]?[0-9]*|([+-]?([0-9]*\.[0-9]+|[0-9]+\.[0-9]*)[Ee][+-]?[0-9]+\.[0-9]+)
```

同样对于 ID_ERROR，定义为数字在前字母在后即可，很简单所以就不贴图了。然后我们对于这些错误，匹配 pattern 和 action，打印出错误信息。这样，我们的词法分析就可以识别出到底是出现了什么错误：

```
carlofkl@ubuntu: $ ./scanner test.cmm
error type A at line 2: mysterious character '36d'
error type A at line 3: mysterious character '35e'
INT: 0
INT: 123
error type A at line 6: mysterious character '0067.6900'
FLOAT: 135.34
error type A at line 8: mysterious character '.56'
error type A at line 9: mysterious character '004e+3'
ID: __int64
ID: A3
ID: a5_56
ID: abc__
TYPE: int
ID: a
error type A at line 14: mysterious character '012389'
TYPE: int
ID: b
INT: 69
TYPE: float
ID: c
FLOAT: 34.23
TYPE: float
ID: c_4
error type A at line 17: mysterious character '002.'
```

test.cmm

```
1 struct
2 36d
3 35e
4 0;
5 123
6 0067.6900
7 135.34
8 .56
9 004e+3
10 __int64
11 A3
12 a5_56
13 abc__
14 int a = 012389,
15 int b = 0x45;
16 float c = 34.23.
17 float c_4 = 002.
```

除了识别出特定的错误以外，我的程序**还能识别出注释符号**，不过在一开始词法分析中，注释符号我是通过正则文法写的，但是这样会在后面的语法分析中出现错误，因为程序虽然会认为该符号合法，但是想”//”这种符号程序很难识别后面到底是注释内容还是程序内容，所以我最后还是选择了用代码实现，即碰到”//”和”/*”之后一次读入字符直至读到”\n”和”*/”。

3. 语法分析

在语法分析部分，做的主要成果在于语法树的建立和错误恢复。程序主要实现的功能包括：

- 构造语法树并按照先序编批的方式打印每一个节点信息。
- 语法单元：打印输入文件中的行号。若产生 e，不打印。
- 词法单元：打印词法单元的名称，不打印该词法单元的行号。
- ID：打印词素；TYPE：打印 int 或 float；NUMBER：打印该十进制数。
- 若程序错误，打印行号，并判断错误类型。并能**进行错误恢复**，让程序继续运行。

在这一部分，我曾经尝试打印语法分析的错误类型，但是这个可能性太多了，所以后来我也没全部写完，就写了一个缺少’]’，但是最后为了整个程序比较统一，也就没使用这部分代码和内容。

首先是语法树的构建，对于一个语法单元或者词法单元，我们声明一个结构体（上文数据结构中提到）。建立的节点的过程就是将 pattern 中匹配到的语法单元和词法单元作为节点内容创建，然后压栈到数组中。

```
/*High-level Definitions*/
Program : ExtDefList {treeNode childList[1] = {$1}; $$= newNode("Program", 1, yylineno, childList);nodeList[nodeNum]=$$;nodeNum++;}
    ;
ExtDefList:ExtDef ExtDefList {treeNode childList[2] = {$1, $2}; $$= newNode("ExtDefList", 2, yylineno, childList);nodeList[nodeNum]=$$;nodeNum++;}
    | {$$= newNode("ExtDefList", 0, -1, NULL);nodeList[nodeNum]=$$;nodeNum++;}
    ;
```

特别的，我们需要判断是否为终结符，若不是则要将儿子数组传入这个节点，若是终结符又要判断是否为 ID，TYPE 或是 INT、FLOAT。如果是这些字符，则将行号令为-1（后续打印节点时候有用），附上部分关键代码：

```
// 有子节点，不是终结符
if (num > 0){
    temp = childList[0];
    setChildTag(temp);
    father -> child[0] = temp;
    father -> line = temp -> line;
    for (int i=1; i<num; i++){
        temp = childList[i];
        setChildTag(temp);
        father -> child[i] = childList[i];
    }
}
```

```
// 终结符、空的语法单元，空单元行号为-1
else{
    father -> line = yyline;
    if ((!strcmp(name, "ID"))) {
        char *str;
        str = (char *)malloc(sizeof(char) * 40);
        strcpy(str, yytext);
        father->ID = str;
    }
```

同时在创造节点的时候，判断是否为终结符（因为是终结符的话必为儿子，但在父节点时候已经打印，后续无需重复打印）。将该节点入栈（以便后续先序遍历的输出），这里栈用数组实现，最后打印的时候只打印非终结符节点的节点内容和他的儿子们（不会造成重复）。特别的，打印的时候只需要打印行号不为-1 的节点且如果他是非终结符，还需要打印行号。

(三) 编译过程

1. 在编译语法分析源代码的时候，利用 bison 进行编译。
2. 编译好的结果会保存在 1.tab.c 和 1.tab.h 这两个文件中，.h 文件中包含了词法单元的类型定义，但是需要在词法分析的源代码中加入对这个.h 文件的引用，然后加了这个引用后再用 flex 编译 1.l 这个源代码。
3. 最后我们得到了 lex.yy.c，当然我们还有 tree.c（主函数），1.h（函数定义和声明）和 1.tab.c（刚刚 bison 的），最后将这些文件链接起来，加入库函数-lfl 和-ly。
4. 最后进行语法分析。

```
carlofkl@ubuntu:~$ bison -d 1.y
carlofkl@ubuntu:~$ flex 1.l
carlofkl@ubuntu:~$ gcc 1.tab.c lex.yy.c tree.c -lfl -ly -o parser
```

(四) 实验结果

仅仅展示一下选做内容的几个结果：

```
carlofkl@ubuntu:~$ ./parser test6.cmm
Error type B at Line 3: syntax error.
Error type B at Line 4: syntax error.
```

```
carlofkl@ubuntu:~$ ./parser test10.cmm
Error type B at Line 7: syntax error.
```

```
carlofkl@ubuntu:~$ ./parser test9.cmm
Program(1)
  ExtDefList(1)
    ExtDef(1)
      Specifier(1)
        TYPE: int
      FunDec(1)
        ID: main
        LP
        RP
      CompSt(2)
        LC
        DefList(7)
          Def(7)
            Specifier(7)
              TYPE: int
            DecList(7)
              Dec(7)
                VarDec(7)
                  ID: i
                ASSIGNOP
                Exp(7)
                  INT: 1
            SEMI
        RC
```

三、 实验总结

1. 学会了如何使用 flex，bison 等工具进行词法和语法的分析。
2. 学会了使用自动机原理，对程序的错误恢复和错误判断有一定的改进。
3. 实现了词法分析中对于八进制、十六进制数、浮点数的错误判断和注释符号识别。