编译原理第一次实验报告

201220069 周心同

实验内容

实现内容：包含所有的必做要求和选做要求

1. 词法分析

根据附录先写出词法单元对应的正则表达式，然后与词法单元进行匹配，匹配正确的创建语法树结点，错误的则输出错误原因，两种情况都要返回属性值给语法分析（避免报重复错误）。

一些特殊形式的正则表达式：

另外，小数点前后必须有数字出现应该是小数点前后之一必须有数字出现

```
int10   [1-9]{digit}*|0
int8    0[0-7]+
int16   0[xX][0-9a-fA-F]+
float   {digit}*\.{digit}+|{digit}+\.
efloat  ({digit}*\.{digit}+|{digit}+\.)[eE][+-]?{digit}+
```

2.语法分析和语法树

根据附录写出正确的产生式， 右边符号属性值全部归为node视作语法树结点，对于每个产生式，左边为创建结点，右边则全部为结点的孩子。对于二义性和冲突处理，根据讲义和附录规定优先级和结合性的方法解决。

语法树就是一个树形结构，每个结点是一个结构体类型，相当于一个语法单元，保存名字，内容，行号，以及子结点和子结点数量。共有三个相关函数，一个createNode初始化一个结点的信息。addNode则是为某个结点加子结点，为了便于在语法分析中添加这里用了可变参数，每个结点都可能有多个子结点，因而这是一个简单的多叉树。printTree则是按照实验要求在正确情况下先序遍历输出对应的信息，注意其中八进制，十六进制和浮点数的转换和空格数量。

3.错误处理

错误有词法错误和语法错误，如果发生错误就不会输出语法树。

type A类型

"/*"：一直input直到遇到“*/”为止，否则就输出错误

int和float中的一些错误形式：通过取补集方式写，优先级如下

```
0[xX][0-9a-zA-Z]*                十六进制
0[0-9a-zA-Z]+                    八进制
[0-9]*[.]|[.][0-9]*              浮点数
[0-9]*[.][0-9]*[Ee][0-9]*        浮点数指数
```

比如十六进制为0[xX][0-9a-zA-Z]*,凡是以0x或0X开头的都视为十六进制形式，而没有成功与前面匹配的自然视为错误的形式。除此之外，将未识别的词法单元全部归为"."即可。

type B类型

通过在语法分析里添加包括error的产生式来找到错误位置，由于实验要求中不要求具体的报错信息，只要求类型和行数，所以可以让yyerror自动报错对应的行数即可。

编译方式

实验环境：与实验要求相同。

在Code目录下make即可。

如果要看自定义的报错信息则需要将syntax.y中的 //#define Error_Recovery中的注释去掉。

程序亮点

增加结点的方式用了可变参数，这样可以减少词法分析的代码量。

部分问题

我通过以下代码输出了部分样例

```
// . {
//  char c = input();
//  while (c !=NULL)
//  {
//      printf("%c",c);
//      c = input();
//  }
// }
```

其中样例e2-2有

```
float f1 = e1.e1
```

该样例进行词法分析时，e1识别为id，但是.e1会被识别为float的错误形式，尽管我已经把'.'的识别规则放在了float错误形式前面，我不明白这是为什么，只能通过去掉float的错误形式识别来解决（当然由于最后 . 的存在这不影响实验结果的判定）