**实验一 词法分析和语法分析**

131220128 杨帆

一、实验环境

操作系统： GNU Linux Release: Ubuntu 12.04

软件版本： GNU Flex version 2.5.35； GNU Bison version 2.5

二、实验内容

编写一个程序对使用C语言书写的源代码进行词法分析和语法分析。若有错误则输出相关信息，否则输出语法树。实验要求使用词法分析工具GNU Flex和语法分析工具GNU Bison，并使用C语言来完成。

三、实验过程

整体思路：

由于编译器的工作特性是先将源程序进行词法分析，然后将识别出的词法单元送入语法分析器，再根据文法分析句子。故必须先完成词法分析的任务。将提取出的token作为参数逐个传入语法分析程序，根据定义的文法规则自底向上进行分析匹配。

为了最后打印语法分析树，必须对终结符、非终结符做建树操作，将每条产生式的左部作父节点，将右部每个符号做子节点（作为兄弟节点用链表连接），打印时对树进行前序遍历即可

完成的功能点：

必做功能   
识别词法错误、识别语法错误

选做功能   
识别八制与十六数、识别指数形式的浮点数(如1.05e-4)、

识别“//”与“/\*…\*/”形式的注释

程序结构与编译过程：

main.c 程序入口。打开文件，启动yystart()分析程序，若无错误则打印语法树。

lexical.l 词法分析程序。定义了词法单元的正则表达式和识别出词法单元后的动作；将终结符号的词法单元建立树节点。

syntax.y 语法分析程序。定义终结符号的优先级，定义合法的文法，建立语法分析树，实现了打印树的过程，打印错误信息。

lex.yy.c 由flex编译lexical.l得到 进行词法分析，被syntax.y调用。

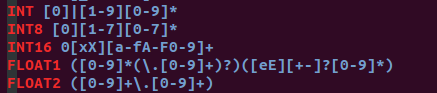
syntax.tab.c syntax.tab.h 由bison编译syntax.y得到，进行语法分析。

bison –d syntax.y flex lexical.l

gcc main.c syntax.tab.c –lfl –ly –o parser

主要功能及其实现方法：

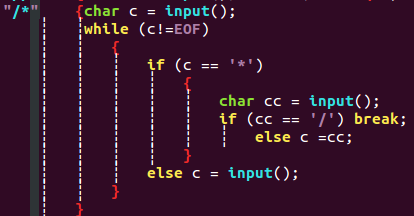
1. 识别8进制、16进制数和无符号浮点数。正则定义如下：



在flex程序的响应函数中，三种整数类型匹配到后都返回到语法分析程序中的INT类型。而FLOAT1、FLOAT2(区分普通浮点数和科学计数法)返回到FLOAT类型。

2.实现识别两种风格的注释

利用flex库函数input(),可以从当前的输入文件中读入一个字符.讲义中已将识别“//”注释行的写1法给出。匹配“/\*…\*/”时while循环直到文件末尾，匹配到一对“/\*”和“\*/”时结束。



3.打印更详细错误信息

使用bison提供的宏 YYERROR\_VERBOSE可以在错误恢复后打印错误信息时先出更详细的内容。例如“unexpected …”“expected …”.

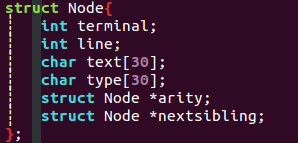
亦可在bison文件定义部分加上%error-verbose.

4.在flex程序中返回给bison程序的内部参数是yylval，表示当前词法单元所对应的属性值，它的默认属性是int类型。而传入bison的数据类型种类很多。我们在定义部分加入union类型，定义为yylval的类型。

%union{struct Node \*node;}

5.语法树的建立与打印

考虑一个节点：每个节点对应待分析程序中的一个符号。节点应该有自己的属性值、属性类型、子节点指针、兄弟节点指针等属性，由于打印时需要打印非终结符号的行号，故还需要保存一个节点的行号。



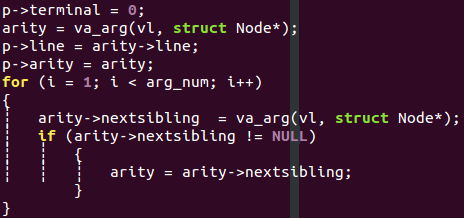
对于终结符来说，它们是整个语法树的叶子节点，故没有子节点。现阶段的终结符节点的建立都在flex程序中识别token后的动作中完成。

对于非终结符，在bison程序中规约到一个非终结符后，执行动作addNode函数，作用是将这个非终结符插入语法树中，并定义这个节点的每个属性。

这里addNode函数，由于每条文法的参数不固定，我们使用c语言提供的库函数stdarg中的va\_list()方法实现不固定多参数的传入，从而实现多叉树的建立。

C:\Users\Micheal\AppData\Roaming\Tencent\Users\2073256873\QQ\WinTemp\RichOle\S)2{9K6KD8QE`J$CU(47G22.png

参数type代表需定义节点的类型、arg\_num代表传入参数的个数



定义了节点p是否为终结符、第一子节点指针、子节点的下一个兄弟节点指针

打印树只需要从深度为0的节点，若遇到的节点是非终结符号节点，打印属性值后继续向下做深度优先遍历。遇到终结符时只需打印其属性即可。由于保存在节点中的属性值是char\*类型（flex内部变量yytext是char\*类型），按实验要求，我们需要将识别出的10进制、8进制、16进制以及浮点数进行字符串到数值类型的转化。此时，常用的atoi()与atof()功能就不够了。通过查阅C手册，使用以下两个转化函数可以实现其功能：

long int strtol(const char \*nptr,char \*\*endptr,int base);

float strtof(const char \*nptr, char \*\*endptr);

6．错误恢复

错误恢复的原理在这里就不再赘述了。通过思考和不断尝试，编写测试样例，在消除shift/reduce conflicts的前提下，在bison定义产生式的部分加入error语句。并重写yyerror（）函数，在分析程序时遇到文法无法匹配的句子时调用它，并打印错误。设置全局变量err，一旦发现一处语句文法无法匹配，就取消打印语法树的过程。

四、实验中的一些注意点

1．在flex程序中 %%rules%% 部分中，定义的顺序决定了分析程序识别token属性的优先级。所以ID的定义必须放在TYPE(int|float)、if、else等正则表达式的后面，否则if、else这样的终结符都会被识别成ID。

bison程序中对于操作符的优先级、左右结合规则参考了实验手册的附录，同时也加入了对悬空else问题、负号和减号的二义性问题的处理。

2.char\*类型的变量无法直接做比较和赋值操作，需要用到string.h库函数：

int strcmp ( const char \* str1, const char \* str2 );

char \* strcpy ( char \* destination, const char \* source );

3.由于本实验是在linux环境下进行的实验，由于编码不同的问题，就无法正确识别其他系统下的文件内容，例如windows下的换行符‘\r’在linux系统下为‘\n’。故在词法分析DELIM的定义中，加入了对‘\r’的识别。

五、实验反思与收获

通过本次实验，我对词法分析以及自底向上的语法分析有了更深层次的理解；提高了对c语言下的工程性代码的编写能力；也为后面的编译器语义分析等部分打下了基础。错误恢复部分仍有不理解和不全面的地方，有待调整和修改。