编译原理实验报告

实验内容：词法分析与语法分析

组长：杨月洋　131220026

组员：林伯宏　131220029

任务编号：17　　联系方式：dxqzyyy@163.com

一、 实现的功能：

对一份输入的以 C--语言书写的源代码，对其进行词法分析，获得词法单元，并根据相关文法进行从底向上的语法分析，最终生成语法树。同时，对于程序中的部分词法和语法错误进行报告。（选做内容也均实现，可通过提供的少许测试样例）。

辅助工具：GNU Flex　GNU Bison

编程语言：C 语言　c99

执行方法：

A. 直接使用目录下的可执行程序 *parser*，输入命令：

*./parser　xx/xx/***.cmm*　(测试文件相对路径)

B. 在 Code 目录下输入 “make”，通过 Makefile 得到可执行文件 *parser*，再进行测试（样例的测试文件均在 Test 目录中）

二、 实现方法：

所有实现代码均在 Code 目录中。除了*”lexical.l”　“syntax.y”　“Makefile”　“main.c”，添加了“type.h”　“pool.c”　“tree.c”　“print.c”*。

在 main 函数中调用 *yyparse()* 开始对一个测试文件进行分析。首

先在 lexical.l 中对字符流进行词法分析，形成一个个词法单元，我在确定生成词法单元的同时，建立该词法对应的“树结点”。

```
CSNode *setCSNode(TokenType t, int no);
CSNode *setCSNode_int(TokenType t, int no, int v);
CSNode *setCSNode_float(TokenType t, int no, float v);
CSNode *setCSNode_id(TokenType t, int no, char *s);
CSNode *setCSNode_relop(TokenType t, int no, char *s);
CSNode *setCSNode_type(TokenType t, int no, char *s);
```

其中 CSNode 为树结点类型，该类型中涵盖了该语法单元的类型，出现的位置，左右指针以及一个表示其类型属性的联合体。建立树结点的方法如图，对于有特定属性值的语法单元，传入相应参数。

当获取了一个词法单元后（开始建立相应终结符的树节点），开始根据 syntax.y 中的文法进行移入和规约操作。

```
ExtDefList:     ExtDef ExtDefList { $$ = setCSNode(MyEXTDEFLIST,@$.first_line);
                                    add2Childs($$,$1,$2);
                                  }
  |     /* empty */                { $$ = NULL; }
  ;
```

每当进行**规约**时，对规约成的非终结符建立树结点。根据从底向上的构造原理，其每个叶子的结点必然已构建完成（所有终结符结点均在词法分析时建立）。然后通过 addXChild(…)方法将其叶子结点依次连入，对于 A -> ε 的产生式，直接将该指针置为 NULL。

规约完成后，最终生成了多叉语法树，我采用的是“左子女右兄弟”的二叉树表示方法，该二叉树的先序遍历与多叉树的先序遍历是相同的。非终结符 Program 作为根节点，用一个 CSNode *root 变量记录其位置，最后根据 root 先序遍历每一个结点，根据不同的语法单元类型输出相应的属性信息。

三、 实验特色：

（1）使用堆区中的静态区域存储结点信息，并使用池结构一次性开辟了特定数目的空间，每次执行建立结点的操作实际只需从池中返回一个指针即可，这样节约了时间，并且便于管理。

```
static PoolNode *p_free = NULL;
#define p_NUM 30
PoolNode *getNode() {
        PoolNode *temp;
        if(p_free == NULL) {
                p_free = (PoolNode *)malloc(sizeof(PoolNode) * p_NUM);
                for(temp = p_free; temp != p_free + p_NUM - 1; temp++) {
                        temp->next = temp + 1;
                }
                temp->next = NULL;
        }
        temp = p_free;
        p_free = p_free->next;
        memset(temp,0,sizeof(PoolNode));
        return temp;
}
```

（当池中的区域被占满时，会自动再次开辟特定数目空间并链接）

（2） 先序遍历“左子女右兄弟”的二叉树时，使用栈替代递归的方式，栈中存储结点指针和结点对应的深度。每次从栈中弹出结点指针，输出其对应的属性信息，并根据弹出的深度 h 给出信息的缩进；然后再顺序压入其兄弟和子女，深度分别为 h 和 h+1。反复操作直至栈为空。（方法为 *tree.c* 中的 *void preOrderTree（CSNode *root）*）

（3） 全局变量 *MY_LEXER_PRINT_FLAG*，默认为 0。在 main 中将其置为 1 后，可以打印每个 lexical.l 中处理的词法单元，之后再输出语法树。当程序出现错误时，若发现打印的内容与源代码不同，即可直接定位是某个词法出错，从而为我的调试过程提供便利。

```
yyy@ubuntu:~/Documents/compiler/remote/Compiler/Lab/Code$ make test
./parser ../Test/test1.cmm
int     main    (       )       \n
{       \n
int     i       =       1(----) ;       \n
int     j       =       2(----) ;       \n
}       \n
******** show the tree *********
Program (1)
  ExtDefList (1)
    ExtDef (1)
      Specifier (1)
        TYPE: int
      FunDec (1)
        ID: main
        LP
        RP
```

识别出的词法