C-Minus 语法分析程序实验报告

姓名：郑明钰
学号：201711210110

一、实验名称：

C-Minus 语法分析程序的设计与实现

二、实验目的：

掌握把 BNF 转换为 EBNF 的方法，再通过递归下降分析方法实现语法分析程序。

即掌握 **EBNF+递归下降分析方法**

三、原理：

（1）C-Minus 语言的上下文无关语法(BNF)：

1. program -> declaration_list
2. declaration_list -> declaration_list declaration | declaration
3. declaration -> var_declaration | fun_declaration
4. var_declaration -> type_specifier ID; | type_specifier ID [NUM];
5. type_specifier -> int|void
6. fun_declaration -> type_specifier ID (params) compound_stmt
7. params -> param_list | VOID
8. param_list -> param_list , param | param
9. param -> type_specifier ID | type_specifier ID [ ]
10. compound_stmt -> { local_declarations statement_list }
11. local_declarations -> local_declarations var_declaration | empty
12. statement_list -> statement_list statement | empty
13. statement -> expression_stmt | compound_stmt | selection_stmt | iteration_stmt | return_stmt
14. expression_stmt -> expression ; | ;
15. selection_stmt ->if (expression) statement | if (expression) statement else statement
16. iteration_stmt -> while (expression) statement
17. return_stmt -> return | return expression
18. expression -> var = expression | simple_expression
19. var -> ID | ID [expression]
20. simple_expression -> additive_expression relop additive_expression | additive_expression
21. relop -> <= | < | > | >= | == | !=
22. additive_expression -> additive_expression addop term | term
23. addop -> + | -
24. term -> term mulop factor | factor
25. mulop -> * | /
26. factor -> (expression) | var | call | NUM
27. call -> ID (args)
28. args -> arg_list | empty
29. arg_list -> arg_list , expression | expression

(2)**BNF 中的左递归会导致编写函数时的无限循环，因此要把 BNF 转换为 EBNF，然后利用 while 和 if 来实现“重复”和“选择”，即：{}和[]**

其中需要修改的地方有：

8.param-list -> param{,param}

11.local-declarations -> empty{var-declaration}

12.statement-list -> empty{statement}

15.selection-stmt ->if (expression) statement [else statement]

20.simple-expression -> additive-expression [relop additive-expression]

22.additive-expression -> term {addop term}

24.term-> factor {mulop factor}

29.arg-list -> expression {,expression}

四、程序的功能

在上一次作业 C-Minus 词法分析程序的基础上，利用扫描程序 scanner 输出的每一个 token 来构建语法树 syntaxTree，然后把语法树 syntaxTree 打印到标准输出流 stdout 中，使其直接显示在命令行中。

五、程序说明

程序相比扫描程序增加了两个文件 parse.h 和 parse.c, 新增代码主要分为三个部分：

语法树结点和语法树定义、实现语法树所需要的实用功能函数、利用递归下降分析方法实现的语法分析部分

（1）定义在 globals.h 中的语法树结点定义和语法树定义

globals.h | scan.c | util.c | main.c | util.h | parse.h | parse.c | scanner.h

```
33  typedef enum
34  {     //共20种语法树结点类型
35      IntK,IdK,VoidK,ConstK,Var_DeclK,Array_DeclK,FunK,ParamsK,Int_ParamK,Array_ParamK,CompK,
36      Selection_StmtK,Iteration_StmtK,Return_StmtK,AssignK,OpK,Array_ElemK,
37      CallK,ArgsK,UnkonwnK
38  }NodeKind;
39
40  typedef enum
41  {
42      Void,Integer
43  }ExpType;
44
45  #define MAXCHILDREN 4   //语法树结点最多有4个孩子，当语法树结点为fun-declaration(FunK)时才需要4个孩子
46
47  typedef struct treeNode
48  {
49      struct treeNode* child[MAXCHILDREN];
50      struct treeNode* sibling;
51      int lineno;   //行号
52      NodeKind nodekind;   //结点类型
53      union
54      {
55          TokenType op;  //存储Op语法树结点的Op类型
56          int val;       //存储ConstK结点的数值大小
57          char* name;    //存储Id结点的Id名字
58      }attr;
59      ExpType type;
60  }TreeNode;
```

（2）位于 util.c 中的实用功能函数如下：

1. char * copyString(char *s)：用于按 token 串的长度复制字符串，节约空间
2. TreeNode* newNode(NodeKind kind)：根据参数构建不同的语法树结点
3. static void printSpaces()：用于打印语法树结点时输出空格
4. void printTree(TreeNode* tree)：打印语法树
5. static int indentno=0;

#define INDENT indentno+=2

#define UNINDENT indentno-=2 ：用于标识此时需要打印的缩进空格的数量，用在 printSpaces()和 printTree 中。

（3）主要位于 parse.c 中的函数，实现递归下降分析的过程,主要函数有:

```
static TreeNode* declaration_list();    //声明列表
static TreeNode* declaration();       //声明：分为变量声明和函数声明
static TreeNode* params();         //函数的参数列表
static TreeNode* param();        //每一个参数
static TreeNode* compound_stmt();     //函数的复合语句
static TreeNode* local_declaration();     //函数内部的局部变量声明
static TreeNode* statement_list();        //语句列表
static TreeNode* statement();            //单独的语句
static TreeNode* expression_stmt();         //表达式语句
static TreeNode* selection_stmt();         //if语句
static TreeNode* iteration_stmt();         //while语句
static TreeNode* return_stmt();        //返回语句
static TreeNode* expression();         //表达式，可以是赋值语句或简单表达式
static TreeNode* simple_expression(TreeNode* k);        //简单表达式
static TreeNode* var();                              //变量
static TreeNode* additive_expression(TreeNode* k);     //可加表达式
static TreeNode* term(TreeNode* k);                 //乘积
static TreeNode* factor(TreeNode* k);             //因子
static TreeNode* call(TreeNode* k);               //函数调用
static TreeNode* args();          //实参列表
```

static TokenType token; 保存当前的 token

TreeNode* parse():语法分析函数，在主函数中调用 parse()完成语法分析，然后用 printTree()打印语法树。

六、输入实例和运行结果：输入文件 test.txt 位于项目文件夹内，三个实例也保存其中，拷贝其内容到 test.txt 即可

1.正确实例 1：

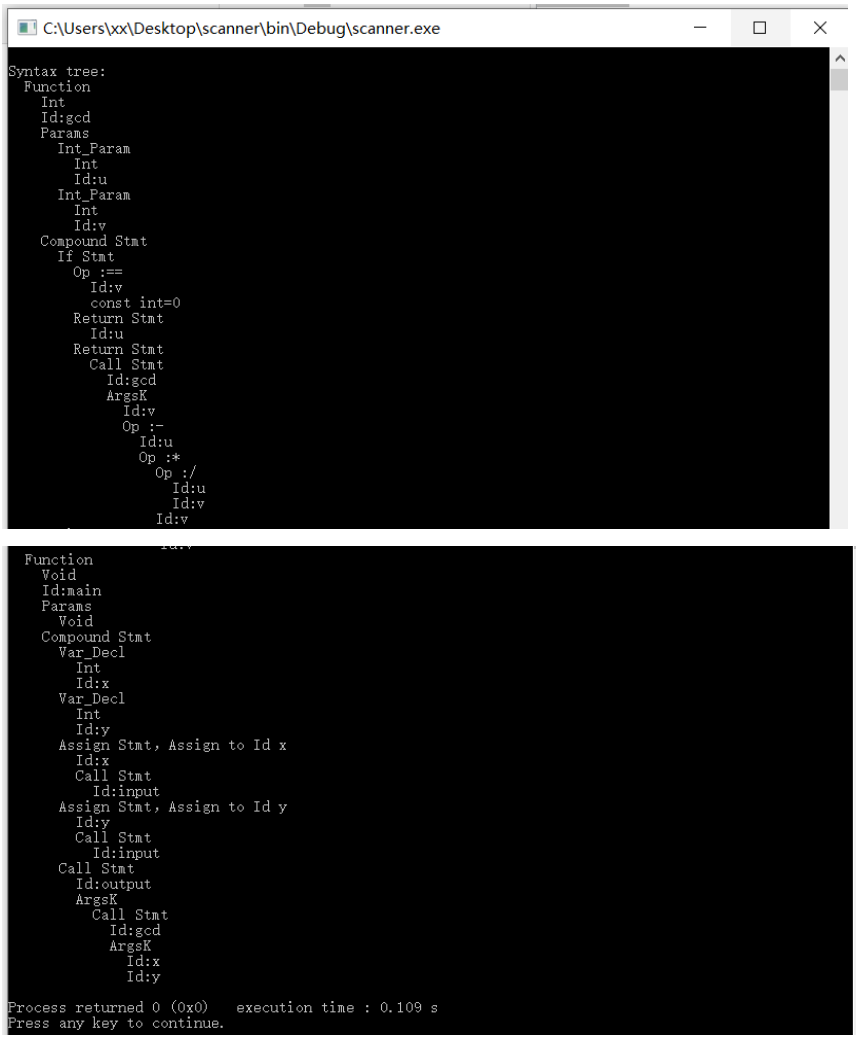
test.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
int gcd (int u,int v)
{
   if (v == 0)
      return u ;
   else
      return gcd(v,u-u/v*v);
   /* u-u/v*v == u mod v */
}

void main(void)
{
   int x; int y;
   x = input();
   y = input();
   output(gcd(x,y));
}
```

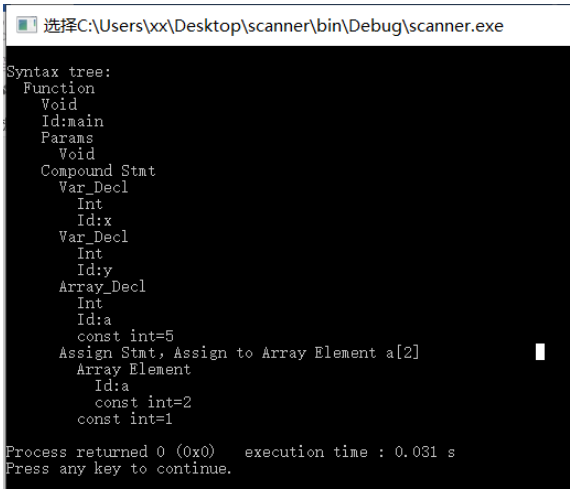
结果如下，输出语法树到 stdout 中，显示在命令行里：



2.正确实例 2:

```
void main(void)
{
    int x; int y;
    int a[5];
    a[2]=1;

}
```



3.错误实例 3：

这个输入实例有两个错误，if 语句和 while 语句的判断条件之后只能再跟一个 statement，是不能像 C 语言那样加上{　}来书写多个 statement 的，这时是不能正确识别的。

```
void main(void)
{
    int a; int b;int c;
    if(a>0)
       {b=1;}
    while(b<0)
          {a=b+c*5-1;}
    return 0;

}
```

```
Syntax tree:
  Function
    Void
    Id:main
    Params
      Void
    Compound Stmt
      Var_Decl
        Int
        Id:a
      Var_Decl
        Int
        Id:b
      Var_Decl
        Int
        Id:c
      If Stmt
        Op :>
          Id:a
          const int=0
        Compound Stmt
          const int=1
      While Stmt
        Op :<
          Id:b
          const int=0
        Compound Stmt
          Op :-
            Op :+
              Id:b
              Op :*
                Id:c
                const int=5
            const int=1
      Return Stmt
        const int=0

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

七、总结：

1.收获：

这次的语法分析器是参考了书后的 Tiny 程序的语法分析器以及网上别人博客的一些思路编写的，明显感觉到代码量远远大于词法分析器。一开始的时候觉得 C-Minus 的语法规则多于 Tiny 的语法规则，怎么才能像书上那样利用递归下降的方法，一个函数一个函数完成语法树构建呢，但是在真正上手开始打之后，发现两者毕竟方法一样，在思路上非常类似，不过具体到 C-Minus 可能需要的函数更多，也需要更加仔细地考虑每条语法规则对应的 follow 集合，从而区分应该属于哪种情况，比如一个以 id 开头的语句到底是赋值语句还是 simple-expression 等。

另外，我还感觉到独自编写代码量较大的程序时，程序各方面的细节考虑的实在太多，需要非常细心，不断尝试新的输入，去发现 bug 然后修改 bug。不过就算这样我想也不能保证程序的健壮性，可能某个未知的 bug 就直接卡死了，连错误信息都没提示，这点我还需要多加努力。总体来说，能够一点点打完语法分析器还是很开心的。

2.遇到的主要问题：

一开始所有应该输出 IdK 的地方都输出的 UnknownK，发现自己在 printTree 的 switch 语句中忘记添加 case:IdK 的情况。

还遇到了表达式 expression 结点不能正确构建的情况，结果发现 bug 出在 factor 中，可见递归下降分析方法在 debug 时候还是非常麻烦的。

3.改进方案：

首先可以一点点地找 bug，不断从细节上完善程序，提高程序遇错处理的能力，另外也可以重新考虑递归下降分析的函数架构，或许有更加清晰的函数设计。