C-Minus 语义分析程序实验报告

姓名：郑明钰
学号：201711210110

一、实验名称：

C-Minus 语义分析程序的设计与实现

二、实验目的：

分析 C-Minus 语言和 Tiny 语言在构造符号表时的不同，在之前语法分析的基础上利用生成的语法树构造符号表

三、原理：

C-Minus 语言和 Tiny 有很大不同，Tiny 语言没有函数，没有局部作用域等，所以在构造语法树时只需要进行一遍前序遍历即可，但是，C-Minus 语言有函数，有局部作用域，所以不能简单地进行一遍前序遍历完成符号表构造。

我的想法是，先**利用一个结构体存储所有的作用域**，这就要修改之前的语法分析部分，在进行递归下降分析时，如果发现进入到一个函数或者局部作用域中，就要添加**当前的作用域**到所有的作用域之中，同时**在语法树结点中增加一个作用域属性**：

```
66    typedef struct scope_list
67    {
68        char *scope_names[50];
69        int current_num;
70    }All_scopes;
71
72    extern All_scopes* Scope_List;
```

（用来存储所有可能的作用域的结构体，假设至多有 50 个可能的作用域）

```
char *scope;                    //作用域属性
```

在构造完语法树之后，对语法树进行前序遍历，如果遇到的是以下结点，则要把对应的标识符 id 添加到符号表中（具体过程在 analyze.c 的 insertNode（）函数）：

1.全局普通变量声明

2.函数声明

3.函数的形参

4.函数的局部变量声明

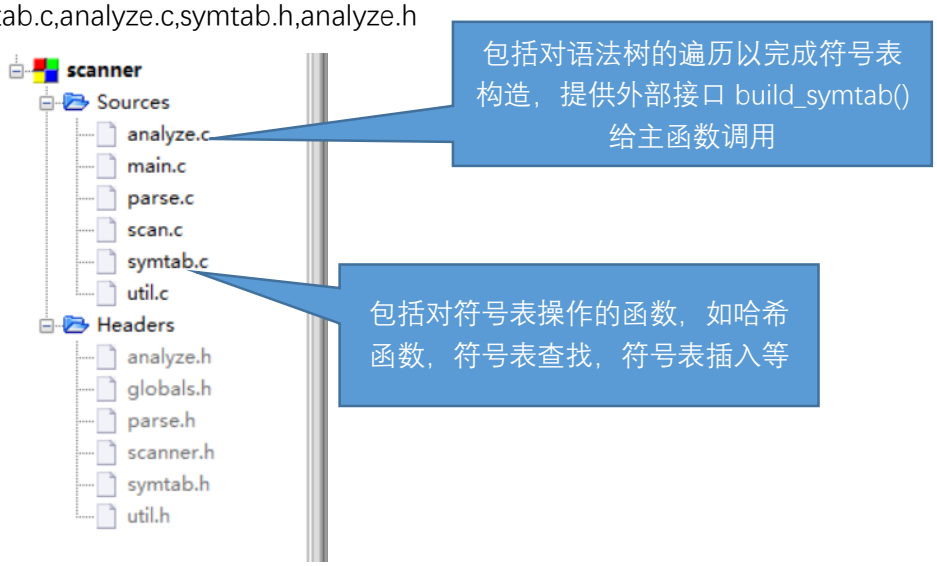
如果遇到的是其他 id 结点，比如表达式中的 id 结点，则从所有可能的作用域里从里往外查找，如果没有找到证明这个变量不符合先声明后使用的原则，如果找到则修改符号表，增加一个新的 line numbers。

四、程序的功能

综合扫描器，语法分析，语义分析的符号表构造部分。从 txt 文件里读入模拟的 C-Minus 程序，生成扫描结果，语法树，符号表。

五、程序说明

（1）程序结构如下：相比之前的语法分析，增加的文件有symtab.c,analyze.c,symtab.h,analyze.h



（2）位于 symtab.c 中的符号表定义如下：

```
typedef struct LineListRec {     // LineListRec
    int lineno; // 记录在源文件的哪一行引用了变量
    struct LineListRec* next;
} * LineList;

typedef struct BucketListRec {  // BucketListRec  哈希表中单个的"桶",哈希表实际上就是一个入口数组
    struct BucketListRec* next; // the next hash
    LineList*   lines;  // reference line  桶对应的链表，存放变量的lineno
    char * name;
    char *  scope;              //变量的作用域，这点和Tiny不同
    TokenType   type;
    int memloc;     // memory location,变量在哈希表中的位置
    int isArr;   //标识是否是数组
}* BucketList;

static BucketList hashTable[SIZE];  //哈希表
```

位于 symtab.c 中的对符号表进行操作的函数定义如下：

1. static int hash(char * key)

//哈希函数，用来计算变量名对应的 hash 值，参考课本 230 页

2. TokenType search_type(char *name,char* scope)

//查找 id 的类型，用于属性检查

3. int st_lookup(char* name,char* scope)

//在符号表里查找变量在内存的存储位置，用于代码生成

4. void printSymTab(FILE* listing)　　//打印符号表

（3）位于 analyze.c 中的函数定义：

1. void insertNode(TreeNode * t)　　//根据语法树的节点不同进行不同的操作
2. static void traverse_for_build_symtab(TreeNode * t)

//用于符号表构造的语法树遍历

3. void buildSymtab(TreeNode* t)　　//给主函数提供的接口

六、输入实例和运行结果：输入文件 test.txt 位于项目文件夹内

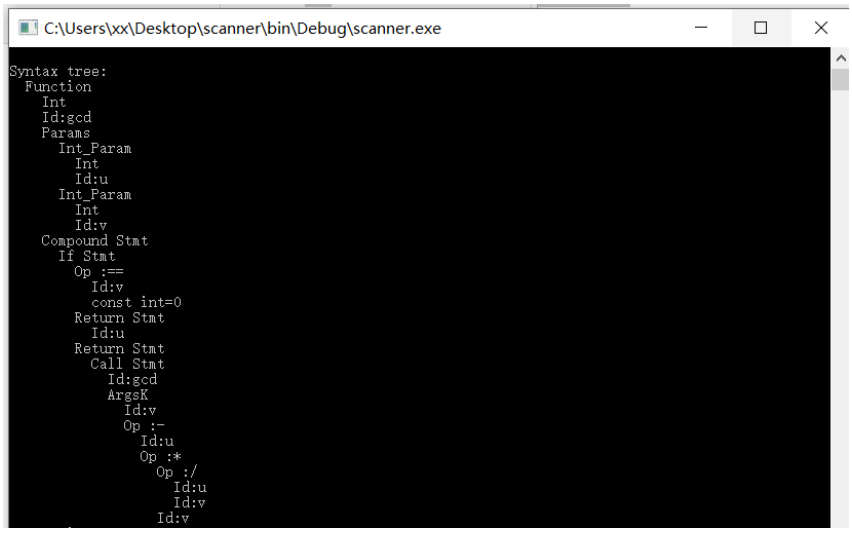
1.正确实例 1：

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：



输出符号表：

```
所有存在的作用域为:
global
gcd
gcd-if_statement0
main
--------------------------------------------------------------------------------

 symbol table of program
Variable Name  Scope            Type    Location    Line Numbers
------------   ------           ------   ------     -----------
u               gcd              int       0      1   4   6   6
v               gcd              int       1      1   3   6   6   6
x               main             int       2     12  13  15
y               main             int       3     12  14  15

Process returned 0 (0x0)   execution time : 0.211 s
Press any key to continue.
```

2.正确实例 2:

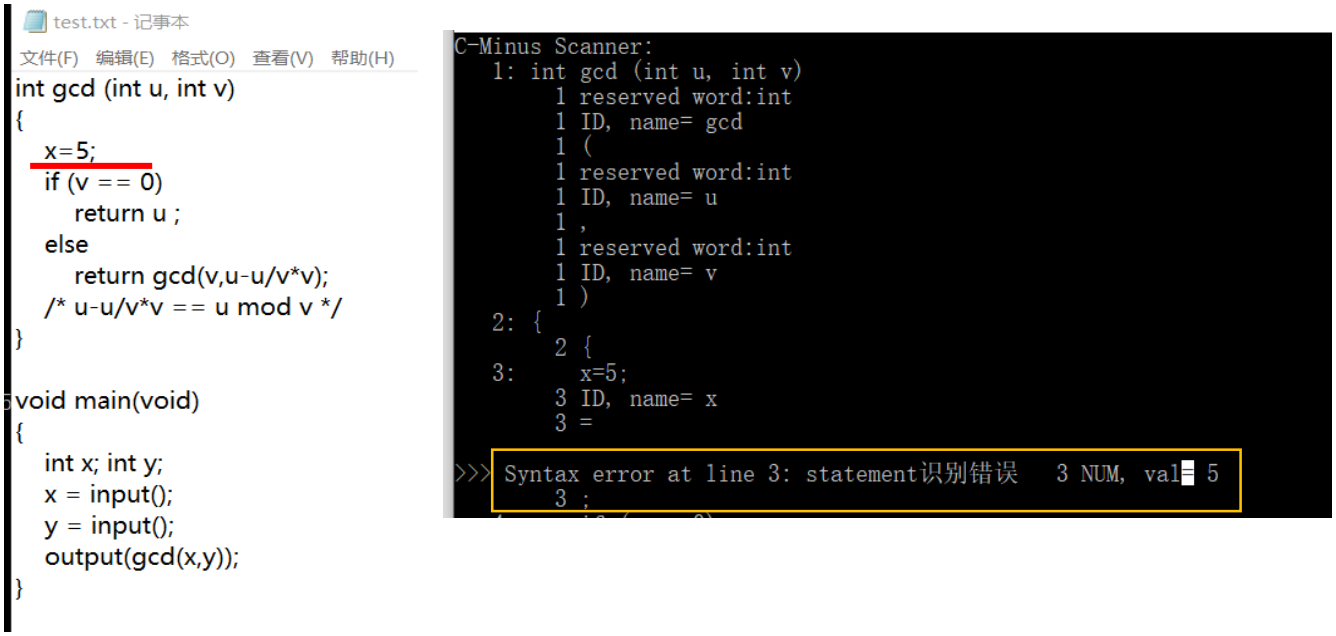
```
void main(void)
{
    int x; int y;
    int a[5];
    a[2]=1;
    
}
```

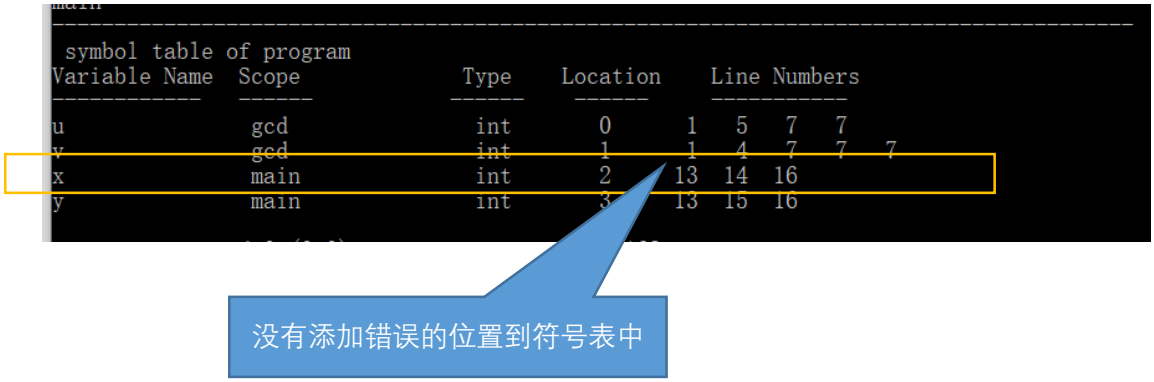
```
选择C:\Users\xx\Desktop\scanner\bin\Debug\scanner.exe

Syntax tree:
  Function
    Void
    Id:main
    Params
      Void
    Compound Stmt
      Var_Decl
        Int
        Id:x
      Var_Decl
        Int
        Id:y
      Array_Decl
        Int
        Id:a
        const int=5
      Assign Stmt, Assign to Array Element a[2]
        Array Element
          Id:a
          const int=2
        const int=1

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

符号表构造如下:

```
所有存在的作用域为:
global
main
--------------------------------------------------------------------------------

 symbol table of program
Variable Name  Scope            Type    Location    Line Numbers
------------   ------           ------   ------     -----------
a               main             int       2      5   6
x               main             int       0      4
y               main             int       1      4
```

3.错误实例 3：x 在使用前没有声明，扫描程序报错，同时不会把这个变量添加到符号表中





七、总结：

1.收获：

通过符号表的构造切实体会到了 Tiny 语言和 C-Minus 语言的不同之处所在，因为 C-Minus 允许函数和局部作用域，使得符号表的构造难度一下子加大了许多，而且这样的符号表的构造只是我自己的想法，肯定还有许多不足之处。同时在考虑问题时也应该把眼光放长远，不能到了语义分析才发现需要返回去修改语法分析程序，否则一旦需要有大的改动实在浪费精力。这点我还需要多加努力。总体来说，能够完全自主完成一个简单的符号表构造还是很开心的。

2.遇到的主要问题：

类型检查部分还未完成，同样的还有代码生成部分。

3.改进方案：

考虑其他更加成熟的符号表构造方法，同时改进程序的遇错处理能力，程序肯定还有很多 bug，如果想得到更加成熟的语义分析程序需要考虑更多的情况，一点点改进。