语法分析设计文档

1. 需求分析

主需求：

基于之前的词法分析程序所识别出的单词，在`testfile.txt`中识别出各类语法成分。输出时，首先按词法分析识别单词的顺序，按行输出每个单词的信息，并指定的语法分析成分分析结束前，另起一行输出当前语法成分的名字。

附加需求：

1. 本作业需要为今后可能出现的错误情况预留接口。

   解决方案：预设一个空的error()函数。

2. 需要设计方便打开/关闭这些输出的方案。

   解决方案：设置一个控制输出全局变量`GRAMMAR_OUTPUT_FLAG`，在输出前用判断语句判断是否输出。

2. 编码前后设计

2.1 自顶而下分析方法

本次语法分析采用的是**自顶而下**的分析方法，给定符号串S，若预测是某一语法成分，则可根据该语法成分的文法，设法为S构造一棵语法树，若成功，则S最终被识别为某一语法成分。自顶而下分析方法有以下的特点：

- 分析过程是带**预测**的，对输入符号串要预测属于什么语法成分，然后根据该语法成分的文法建立语法树；
- 分析过程是一种**试探过程**，是用尽一切办法（选用不同规则）来建立语法树的过程，由于是试探过程，难免有失败，所以分析过程需进行回溯，因此也称这种方法是**带回溯的自顶而下分析方法**。

自顶而下分析的缺点是不能处理具有**左递归性**的文法，因此我们需要首先检查文法是否具有左递归性，若有，则需要进行消除。但我们的文法不存在左递归性，因此可以带回溯的自顶而下分析方法。

针对回溯问题，采用的是**超前扫描**（偷看）的方法。当文法不满足避免回溯的条件时，即各选择的首符号相交时，可以采用超前扫描的方法，即向前侦察各输入符号串的第二个、第三个符号来确定要选择的目标。

在超前扫描的具体实现上，**编码前后有所不同**：

- 编码前：将预读取到的第二个符号、第三个符号等符号用类似`token_2`、`token_3`的命名方式的变量进行存储，然后从后往前存入名为`token_stack`的栈中。当语法分析程序想得到下一个字符，调用`getsym()`时，如果`token_stack`非空，就优先读取`token_stack`中弹出的符号。
- 编码后：原先的方式压栈时语句太过冗余，使代码的可读性大幅降低。因此，在编码后，用两个栈的来进行超前扫描。首先，将超前扫描得到的符号压入临时栈`temp_token_stack`，再超前扫描结束后，再用自定义的`move_stack()`函数依次将`temp_token_stack`中的元素压入`token_stack`栈中。这样的话语句简洁，可读性好。

2.2 递归下降分析法

递归下降分析法也称递归子程序法，具体做法是对语法的每一个非终结符都编一个分析程序，当根据文法和当时的输入符号预测到要用某个非终结符去匹配输入串时，就调用该非终结符的分析程序。其中，命名方式如下所示。

非终结符号	分析程序名
<程序>	program
<有返回值函数定义>	func_def_with_ret
<声明头部>	declaration_head
<无返回值函数定义>	func_def_no_ret
<参数表>	para_tlb
<主函数>	main_func
<复合语句>	compound_statements
<语句列>	statement_list
<常量说明>	const_declaration
<常量定义>	const_def
<变量说明>	var_declaration
<变量定义>	var_def
<变量定义无初始化>	var_def_no_init
<变量定义及初始化>	var_def_with_init
<语句>	statement
<循环语句>	loop_statement
<步长>	step
<条件语句>	condition_statement
<条件>	condition
<关系运算符>	relation_op
<有返回值函数调用语句>	func_call_with_ret
<无返回值函数调用语句>	func_call_no_ret
<值参数表>	value_para_tlb
<赋值语句>	assign_statement
<读语句>	input_statement
<写语句>	output_statement
<情况语句>	case_statement
<情况表>	case_tlb
<情况子语句>	case_substatement
<缺省>	default_statement

非终结符号	分析程序名
<返回语句>	ret_statement
<表达式>	expr
<项>	term
<因子>	factor
<类型标识符>	type_ident
<标识符>	ident
<字符串>	string_sym
<常量>	const_sym
<字符>	character
<加法运算符>	add_op
<乘法运算符>	multi_op
<字母>	letter
<整数>	integer
<无符号整数>	unsigned_integer
<数字>	number

所有的分析子程序统一约定， 在编写程序时，要求进入某个非终结符的分析子程序前，必须先将所要分析的语法成分第一个符号读入`token`中；而在分析子程序的出口前，一定要读取下一个符号到`token`中，以便为进入下一个分析子程序做好准备。

此处，在具体实现上，采用的是**基于递归下降分析法的语法分析程序构造**，编码前后设计大致相同，在一些细节上略有微调。例如，在<语句>的分析子程序中，需要区分是<有返回值函数调用语句>还是<无返回值函数调用语句>，所以需要建立一个有返回值函数集合和无返回值函数集合，在<有返回值函数定义>和<无返回值函数定义>时，分别存入相应的集合。最初设计时，是在整个函数定义没有出现错误，在分析子程序的末尾才加入集合；但是，有在函数定义就调用该函数，采用原先的设计方案就会报错，所以需要在出现该函数名时就加入函数集合。