# 语法分析阶段设计文档



## 一、编码之前的设计

语法分析的主要任务是根据文法规则，从源程序单词符号串中识别出语法成分，并进行语法检查。

在本次作业中，我的程序使用的主要思路是**自顶向下的递归子程序法。**具体来说，就是给语法的每一个非终结符都编写一个分析程序，当根据文法和当前的输入符号预测到要用到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。

本次作业的文法规则中没有左递归的文法，但是部分非终结符的规则右部的多个选择会有一些相交的首符号集合。因此，遇到这些非终结符时（例如<语句>）我在本次作业中使用了**超前扫描**的方法来规避回溯的操作。

关于词法分析程序和语法分析程序的关系，我这里采用了**词法分析程序作为单独的子程序**的实现方案。在分析语法成分的同时不断从词法分析子程序中取单词，从而提高程序的效率。

## 二、编码之后对设计的修改

### 函数和变量说明

以下是我在本次作业中使用到的函数。

```
// 以下为需要使用的外部函数，主要是词法分析的相关函数和错误处理函数
extern void getsym();                // 输出当前的单词，并获取下一个单词
extern void trysym();                // 仅获取当前的单词，用于超前扫描
extern void ungetsym();              // 回退一个单词，用于超前扫描
extern void outputsym();             // 输出当前的单词
extern void outputlastsym();         // 输出上一个单词
extern void error();                 // 错误处理函数

// 以下为分析语法成分的递归子程序（按照文法定义的顺序排列）
void getAdditionOperator();          // <加法运算符>
void getMultiplicationOperator();    // <乘法运算符>
void getRelationalOperator();        // <关系运算符>
void getLetter();                    // <字母>
void getDigit();                     // <数字>
void getChar();                      // <字符>
```

```
void getString();                    //*<字符串>

void getProgram();                   //*<程序>
void getConstantDeclaration();       //*<常量说明>
void getConstantDefination();        //*<常量定义>
void getUnsignedInteger();           //*<无符号整数>
void getInteger();                   //*<整数>

void getIdentifier();                // <标识符>
void getDeclarationHeader();         //*<声明头部>
void getConstant();                  //*<常量>
void getVariableDeclaration();       //*<变量说明>
void getVariableDefination();        //*<变量定义>
void getVariableDefinationWithoutInitialization();  //*<变量定义无初始化>
void getVariableDefinationWithInitialization();     //*<变量定义及初始化>
void getTypeIdentifier();            // <类型标识符>

void getFunctionDefinationWithReturnValue();    //*<有返回值函数定义>
void getFunctionDefinationWithoutReturnValue(); //*<无返回值函数定义>
void getCompoundStatements();                   //*<复合语句>
void getParameterList();                        //*<参数表>
void getMain();                                 //*<主函数>

void getExpression();   //*<表达式>
void getTerm();         //*<项>
void getFactor();       //*<因子>

void getStatement();
void getAssignmentStatement();
void getConditionalStatement();
void getCondition();
void getLoopStatement();
void getStep();
void getSwitch();
void getCaseTable();
void getCase();
void getDefault();
void getFunctionCallStatementWithReturnValue();
void getFunctionCallStatementWithoutReturnValue();
void getValueParameterTable();
void getStatementList();
void getScanf();
void getPrintf();
void getReturn();

// 以下为 getProgram 函数中需要使用到的函数
void getVariableDeclarationForProgram();
void getVariableDefinationForProgram();
void getMainForProgram();
void getFunctionDefinationWithReturnValueForProgram();
void getFunctionDefinationWithoutReturnValueForProgram();
```

以下是我在本次作业中使用到的变量。

```cpp
// 以下三个为来自外部的变量
extern ofstream ofile;          // 输出文件流
extern int symbol;              // 记录当前所识别单词的类型
extern char TOKEN[256];         // 记录当前所识别单词的字符串

enum classnum {IDENFR, INTCON, CHARCON, STRCON, CONSTTK, INTTK, CHARTK,
VOIDTK, MAINTK, IFTK, ELSETK, SWITCHTK, CASETK, DEFAULTTK, WHILETK, FORTK,
SCANFTK, PRINTFTK, RETURNTK, PLUS, MINU, MULT, DIV, LSS, LEQ, GRE, GEQ,
EQL, NEQ, COLON, ASSIGN, SEMICN, COMMA, LPARENT, RPARENT, LBRACK, RBRACK,
LBRACE, RBRACE};        // 单词类别编码
int dimension = 0;                          // 保存变量定义的数组维数
string lastIdentifier;                      // 保存识别的上一个单词的标识符
set<string> functionWithReturnValue;    // 保存有返回值函数函数名的set容器
set<string> functionWithoutReturnValue; // 保存无返回值函数函数名的set容器
```

# 设计思路说明

## 第零步 预处理

打开相应的输入流文件和输出流文件，然后初始化词法分析的内部编码。

## 第一步 预读取一个单词并开始 getProgram() 函数

在我们的文法规则中，根节点是`<程序>`这个语法成分，因此我们的所有语法分析都是从`getProgram()`这个函数开始的。

在预读取一个单词之后，我们可以在`<程序>`的右部中寻找可能成立的选择，并根据不同选择的前部和预读取的单词类别进行匹配，进而调用递归子程序进行下一步的扫描。

以下是我的主函数。

```cpp
int main() {
    ifile.open("testfile.txt", ios::in);
    ofile.open("output.txt", ios::out);

    init_CODEN();
    init_CODES();

    // 词法分析和语法分析
    getsym();
    getProgram();

    ifile.close();
    ofile.close();
    return 0;
}
```

## 后续步骤 不断调用递归子程序

在调用了`getProgram()`函数后，我们要做的就是不断地根据文法规则调用递归子程序。直至词法分析得到的所有单词都能根据文法规则找到对应的语法成分为止。

本次作业虽然在思路上并不复杂，然而，主要难度在于众多的语法成分以及部分语法成分的具有相同首符号的右部。前者需要我们注意细节从而减少错误的发生，后者则需要我们使用超前扫描的方法，确定具体的语法成分。

例如对于`<程序>`右部的`<变量说明>`和`<有返回值函数定义>`这两个的语法成分，具有相同的首符号，均为`int`或者`char`，同时第二个符号也都是标识符。因此，我们需要至少超前扫描两个单词才能具体确认右部匹配的语法成分。就这个问题而言，当第三个语法成分为左括号时代表的是`<有返回值函数定义>`，反之则是`<变量说明>`。以下是关于这个部分的代码。

```
while (symbol == INTTK || symbol == CHARTK) {
    trysym();       // 超前扫描一个单词
    trysym();       // 超前扫描一个单词
    if (symbol != LPARENT) {
        hasVariable++;
        ungetsym();
        outputlastsym();
        getVariableDeclarationForProgram();
    }
    else {
        ungetsym(); // 回退一个单词
        ungetsym(); // 回退一个单词
        break;
    }
}
```

在根据每个语法成分都编写了递归子程序之后，我们就可以从`<程序>`这个根结点出发将语法分析子程序扫描到的单词与文法定义中的非终结符一一匹配，从而完成语法分析的整个过程。