词法分析阶段设计文档

一、编码之前的设计

在编码之前， 我大致思路是首先是按单词种类进行分类，对保留字和分界符采用一符一类的方法，之后根据状态图进行不同种类单词的识别。简单来说，就是根据词法规则分析出状态图，然后根据状态图进一步写出词法分析程序。

以下是我们本次作业要求的单词和相应的类别码。

单词名称	类别码	单词名称	类别码	单词名称	类别码	单词名称	类别码
标识符	IDENFR	else	ELSETK	-	MINU	=	ASSIGN
整形常量	INTCON	switch	SWITCHTK	*	MULT	;	SEMICN
字符常量	CHARCON	case	CASETK	/	DIV	,	COMMA
字符串	STRCON	default	DEFAULTTK	<	LSS	(	LPARENT
const	CONSTTK	while	WHILETK	<=	LEQ	)	RPARENT
int	INTTK	for	FORTK	>	GRE	[	LBRACK
char	CHARTK	scanf	SCANFTK	>=	GEQ	]	RBRACK
void	VOIDTK	printf	PRINTFTK	==	EQL	{	LBRACE
main	MAINTK	return	RETURNTK	!=	NEQ	}	RBRACE
if	IFTK	+	PLUS	:	COLON		

二、编码之后对设计的修改

在确定了理论上的大致思路以及要求完成的单词种类之后，我们就可以进行代码相关的具体分析。

函数和变量说明

接下来，我们来看一下本次作业中用到的所有函数和变量，具体的含义均标注在了注释中。

以下是我在本次作业中使用到的所有函数。

```
int isDigit(char c);          // 判断字符是否为数字
int isLetter(char c);         // 判断字符是否为字母
int isChar(char c);           // 判断是否为字符
int isString(char c);         // 判断是否为字符串中的字符
void error();                 // 错误处理函数
void cat(char s[], char c);   // 将字符插入到字符串的末尾
void init_CODEN();            // 初始化单词类别编码
void init_CODES();            // 初始化单词对应的字符串
void getsym();                // ***获取下一个 token
int getnbc();                 // 从字符缓冲区获取下一个非空字符，返回 1 表示成功，返回 0 表示失败
void getch();                 // 从字符缓冲区获取一个字符
void ungetch();               // 从字符缓冲区回退一个字符
void reserve(char s[]);       // 判断获取到的字符串是否为保留字
void tolowercase(char s[]); // 将字符串中的大写英文字符转化为小写
```

以下是我在本次作业中使用到的所有变量。

```
char CHAR;                    // 存放当前读入的字符
char TOKEN[256];              // 存放单词字符串
char buffer[256];             // 文件输入缓冲区
int symbol;                   // 保存当前所识别单词的类型
int cnt = 0;                  // 记录下一个读取的字符的位置
int cnt_len = 0;              // 记录当前缓冲区字符串的长度
int cnt_line = 0;             // 记录当前缓冲区在源文件中的行数，以便进行错误输出
int num = 0;                  // 存放当前读入的整型数值
string CODEN[40];             // 保存单词类别编码
string CODES[40];             // 保存单词对应字符串
```

设计思路说明

第零步 预处理

在预处理阶段，创建了`ifstream`和`ofstream`分别对应输入文件和输出文件，同时对`CODEN`和`CODES`数组进行了初始化，为后续的处理做了准备。

在明确了单词的类别码之后，我对类别码分配了内部编码，并将类别码保存在了一个string数组中。以下是我的保存类别码的数组的初始化函数。该数组的下标对应的就是该类单词的内部编码。

```
void init_CODEN() {
    CODEN[0] = "IDENFR";        // 标识符
    CODEN[1] = "INTCON";        // 整型常量
    CODEN[2] = "CHARCON";       // 字符常量
    CODEN[3] = "STRCON";        // 字符串
    CODEN[4] = "CONSTTK";
    CODEN[5] = "INTTK";
    CODEN[6] = "CHARTK";
    CODEN[7] = "VOIDTK";
    CODEN[8] = "MAINTK";
    CODEN[9] = "IFTK";
    CODEN[10] = "ELSETK";
    CODEN[11] = "SWITCHTK";
    CODEN[12] = "CASETK";
    CODEN[13] = "DEFAULTTK";
```

```
    CODEN[14] = "WHILETK";
    CODEN[15] = "FORTK";
    CODEN[16] = "SCANFTK";
    CODEN[17] = "PRINTFTK";
    CODEN[18] = "RETURNTK";
    CODEN[19] = "PLUS";
    CODEN[20] = "MINU";
    CODEN[21] = "MULT";
    CODEN[22] = "DIV";
    CODEN[23] = "LSS";
    CODEN[24] = "LEQ";
    CODEN[25] = "GRE";
    CODEN[26] = "GEQ";
    CODEN[27] = "EQL";
    CODEN[28] = "NEQ";
    CODEN[29] = "COLON";
    CODEN[30] = "ASSIGN";
    CODEN[31] = "SEMICN";
    CODEN[32] = "COMMA";
    CODEN[33] = "LPARENT";
    CODEN[34] = "RPARENT";
    CODEN[35] = "LBRACK";
    CODEN[36] = "RBRACK";
    CODEN[37] = "LBRACE";
    CODEN[38] = "RBRACE";
}
```

第一步 从文件中读取到缓冲区

我使用了C++的 `<fstream>` 来进行文件的读取。通过循环不断将文件中的一行代码传入到程序的输入缓冲区，也就是 `buffer` 数组中，然后调用 `getsym()` 函数不断获取单词并输出到输出文件直到该行的每一个字符都已经被扫描过。然后再读取文件的下一行代码进行同样的处理。

以下是这一部分相关的代码。

```
while (!ifile.eof()) {
    // 初始化相关变量
    cnt = 0;
    cnt_line++;
    // 从文件中读取一行字符串
    ifile.getline(buffer, 256);
    cnt_len = strlen(buffer);
    // 对该行字符串进行分析
    while (cnt < cnt_len) {
        getsym();
        outputsym();
    }
}
```

第二步 从缓冲区获取单词

这是本次作业最重要的部分，主要涉及的是 `getsym()` 函数，目标就是从输入缓冲区中获取下一个单词。

我设置了三个跟读取指针相关的函数，用于从字符缓冲区获取下一个字符。

- `int getnbc();` 从字符缓冲区获取下一个非空字符
- `void getch();` 从字符缓冲区获取一个字符
- `void ungetch();` 从字符缓冲区回退一个字符

根据此次词法分析作业的要求，单词主要可以分为五类。

- 以<字母>开始的，包括标识符、const、int……return等
- 以<数字>开始的，主要是整形常量
- 以单引号开始的字符常量
- 以双引号开始的字符串
- 以特殊字符开始的特殊符号

在使用`getnbc()`函数获取了下一个非空字符之后，我们就可以确定单词的大致种类。然后按照上述五类的单词根据文法要求进行进一步的分析处理。

第三步 输出获取到的单词

我将`getsym()`函数的返回值设置为了内部编码，也就是每个单词名称对应的`CODEN`数组下标。在主函数中，通过收到的返回值按照不同的单词种类进行输出。

单词的输出也分两种情况，倘若是特殊字符，则输出`CODES`数组中保存的对应的名称；其余情况则输出`TOKEN`变量中保存的字符串。这么做的主要原因是部分的保留字会出现大写，例如本次作业的样例中的`CONst`，这导致不能通过内部编码直接输出预设的字符串，而必须输出程序运行过程中实际读取到的字符串。