词法分析设计文档

1. 需求分析

主需求：

从`textfile.txt`中的测试程序识别出单词，并记录其单词类别和单词值，并将词法分析结果以`单词类别码 单词的字符/字符串形式`输出至`output.txt`。

附加需求

1. 读取的字符串要原样保留，输出的不是真正的单词值，而是读入的字符串，单词值需另行记录；

   解决方案：设计一个`token`类，每个单词都是`token`类的对象，其中包含了类别码`wd_type`，原样字符串`wd_string`，单词值`wd_value`等属性。

2. 需要为今后可能出现的错误情况预留接口；

   解决方案：当返回的`token`的`wd_type`是负数时，说明出现了错误情况。

3. 之后在错误处理中，需要输出错误的行号；

   解决方案：token类的属性中添加单词所处的行号，同时在输入时要逐行逐行输入，以便记录行号。

4. 需要设计方便打开/关闭这些输出的方案；

   解决方案：设置一个控制输出的全局变量`LEXER_OUTPUT_FLAG`，在输出前用if进行判断是否输出。

2. PL/0编译器和Pascal-S编译器词法分析

2.1 PL/0词法分析程序getsym

getsym作为一个独立的子程序（过程）由语法分析程序调用，它的主要功能如下：

（1）**跳过源程序中的空格字符**；

（2）**从源程序正文字符序列中识别出单词符号，并把该单词符号类别以相应枚举值的形式（即内部编码）送入变量`sym`中**；

（3）**用变量id存放标识符，用二分法查找保留字表，识别诸如begin、end等保留字。**

```
procedure getsym;
    var i,j,k : integer;
    begin { procedure getsym; }
    while ch = ' ' do getch;
    if ch in ['a'..'z'] then begin { identifier of reserved word }
        k := 0;
        repeat
            if k < al then begin
                k := k+1;
                a[k] := ch
            end;
            getch
        until not( ch in ['a'..'z','0'..'9']);
        if k >= kk { kk : last identifier length } then
```

```
            kk := k
        else
            repeat
                a[kk] := ' ';
                kk := kk-1
            until kk = k;
        id := a;
        i := 1;
        j := norw; { binary search reserved word table }
        repeat
            k := (i+j) div 2;
            if id <= word[k] then j := k-1;
            if id >= word[k] then i := k+1
        until i > j;
        if i-1 > j then sym := wsym[k] else sym := ident
    end
    else
        if ch in ['0'..'9'] then begin { number }
            k := 0;
            num := 0;
            sym := number;
            repeat
                num := 10*num+(ord(ch)-ord('0'));
                k := k+1;
                getch
            until not( ch in ['0'..'9']);
            if k > nmax then error(30)
        end
        else
            if ch = ':' then begin
                getch;
                if ch = '=' then begin
                    sym := becomes;
                    getch
                end
                else sym := nul
            end
            else
                if ch = '<' then begin
                    getch;
                    if ch = '=' then begin
                        sym := leq;
                        getch
                    end
                    else
                        if ch = '>' then begin
                            sym := neq;
                            getch
                        end
                        else
                            sym := lss
                end
                else
                    if ch = '>' then begin
                        getch;
                        if ch = '=' then begin
                            sym := geq;
                            getch
```

```
                        end
                        else
                            sym := gtr
                    end
                    else begin
                        sym := ssym[ch];
                        getch
                    end
    end; { getsym }
```

`getsym`调用`getch`扫描输入的源程序，取来一个个字符。`getch`实际上是将**输入文件中的每一行源程序先读入缓冲区line中，然后再从line中取出字符。程序用变量ll记录当前源程序行的长度，用变量cc对当前所取字符在该行中的位置进行计数。**同时`getsym`还需要完成：

（1）**识别并越过行结束信息**；

（2）把从input文件读入的源程序同时输出到output文件，已形成被编译源程序的清单（用户可以在终端屏幕上看到编译扫描的进程）（DEBUG时可用）；

（3）在输出的每一条源程序的开始处打印出编译生成的目标指令行号。

```
procedure getch;
begin
    if cc = ll then begin { get character to end of line }
        { read next line }
        if eof(fin) then begin
            writeln('program incomplete');
            close(fin);
            exit;
        end;
        ll := 0;
        cc := 0;
        write(cx:4,' '); { print code address }
        while not eoln(fin) do begin
            ll := ll+1;
            read(fin,ch);
            write(ch);
            line[ll] := ch
        end;
        writeln;
        readln(fin);
        ll := ll+1;
        line[ll] := ' ' { process end-line }
    end;
    cc := cc+1;
    ch := line[cc]
end; { getch }
```

2.2 Pascal-S词法分析程序insymbol

词法分析程序主要由`nextch`（取字符）、`insymbol`（取单词）、`readscale`（读指数）和`adjustscale`（求出实数值）等过程组成，其大部分动作与PL/0相似。

3. 编码前设计

3.1 读入字符

`InputFile`

`InputFile.h`中定义一个`Inputfile`类，具体类成员和类成员函数的属性及含义如下所示。

类成员名	访问修饰符	类型	含义
file_path	private	string	源程序的地址
cur_col	private	int	当前正在处理的字符的列号
cur_line	private	int	当前正在处理的字符的行号
cur_line_code	private	string	当前正在处理的行的代码
infile	private	ifstream	输入文件流
len_lines	private	list	记录源程序每行的长度

类成员函数名	类访问修饰符	输入参数	返回类型	含义
InputFile	public	string file_path	void	构造函数
~InputFile	public	void	void	析构函数
getFilePath	public	void	string	返回源程序的地址
getInfile	public	void	ifstream	返回输入文件流
getch	public	void	char	读取单个字符
getLineLen	public	int line	int	返回源程序第line行的长度
getNextLine	public	void	string	读源程序下一行

3.2 单词识别

编码前单词识别设计与编码后相同，详见4.2。

4. 编码后设计

4.1 读入字符

`InputFile`

`InputFile.h`中定义`InputFile`类，具体类成员和类成员函数的属性及含义如下所示。

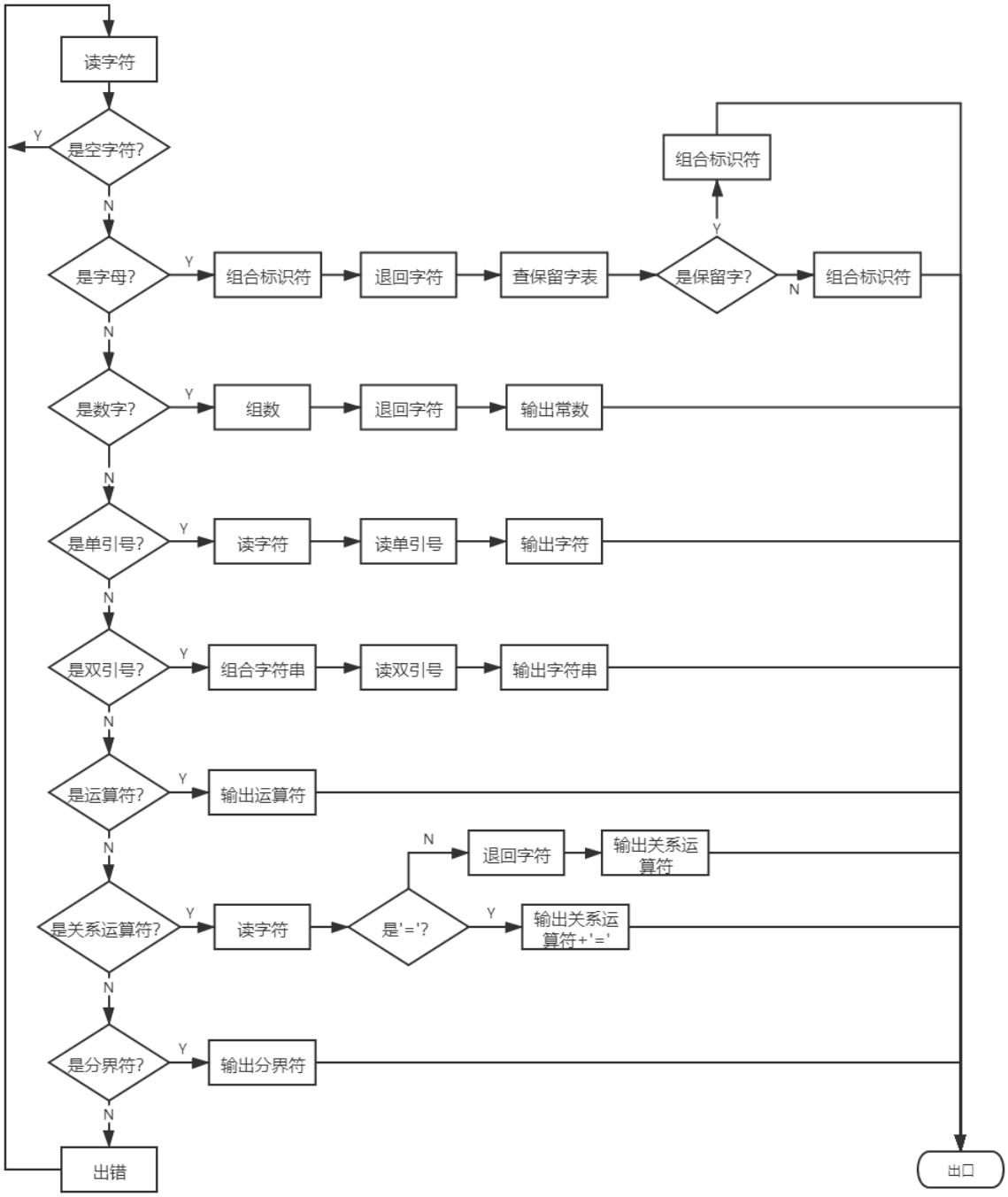
类成员名	访问修饰符	类型	含义
file_path	private	string	源程序的地址
cur_col	private	int	当前正在处理的字符的列号
cur_line	private	int	当前正在处理的字符的行号
cur_line_code	private	string	当前正在处理的行的代码
len_lines	private	map	存储源程序每一行的长度
infile	public	ifstream	输入文件流

- 由于C++中不能返回`ifstream`类型的变量，所以把`infile`类型改为`public`。
- C++中`list`是以链表的形式进行存储，每次需要得到源程序某一行的长度时，都需要从第1行开始遍历，因此`len_lines`采用map类型，更加符合我们的需求。

类成员函数名	类访问修饰符	输入参数	返回类型	含义
InputFile	public	string file_path	void	构造函数
~InputFile	public	void	void	析构函数
getFilePath	public	void	string	返回源程序的地址
getch	public	void	pair<char, int>	读取单个字符及字符的行号
getLineLen	public	int line	map	返回源程序第line行的长度
retract	public	void	void	指针回退一个字符

- 不需要`getNextLine`来得到下一行的代码，用`ifstream`类中封装好的函数`getline`就可以
- 因为在之后错误处理时，需要返回行号，所以`getch`的返回类型从`char`增加到了`pair<char, int>`
- 需要添加`retract`函数，当多读取一个字符时，指针回退一个字符。

4.2 单词识别


读字符
是空字符?
Y
N
是字母?
Y
组合标识符
退回字符
查保留字表
是保留字?
Y
N
组合标识符
是数字?
Y
组数
退回字符
输出常数
是单引号?
Y
读字符
读单引号
输出字符
是双引号?
Y
组合字符串
读双引号
输出字符串
是运算符?
Y
输出运算符
是关系运算符?
Y
读字符
是'='?
N
退回字符
输出关系运算符
Y
输出关系运算符+'='
是分界符?
Y
输出分界符
出错
出口