**LD语言的数据结构以及应用场景**

作者：韩美卿

日期：2015.11

**摘要:**

IEC61131-3国际标准的编程语言包括图形化编程语言和文本化编程语言。LD（Ladder Diagram）语言是编程中被最广泛使用的一种图形化语言。本文结合DCS\_AT项目来介绍LD语言的设计流程，着重讲解其设计主要涉及到的数据结构，并最后结合实例介绍该语言主要的应用场景。

**关键词：**

LD 数据结构 应用场景

### 一、LD语言介绍

LD语言是采用类似于继电器梯形逻辑图的“梯级”方式的符号在网络中布局，来表达控制中的节点当前通过的电流。它包含一系列网络（节）。网络通过左右垂直线被限制在左边和右边。在中间是由触点、线圈和连接线组成的电路图。

#### LD语言在开发项目中的位置

图 1-1 系统结构图

LD语言模块在系统中的位置用深色字体表示。

#### LD语言模块设计

LD语言模块按照功能相对独立性可以划分为：LD编辑器、LD语法检查和LD逻辑转换三个部分。

LD编辑器是用于用户使用LD图形元素进行逻辑组态，它提供一系列的基本元素类（触点、线圈、功能块、节等），并支持各种编辑操作（插入、删除、撤销恢复、查找替换等等），它也是LD语法检查和逻辑转换的基础，本文结合DCS\_AT项目详细讲述LD编辑器的各种图形元素类的结构以及主要数据结构设计。

### 二、主要数据结构设计

#### 1. 文档视图结构图

图2-1 文档视图结构图

#### 2. LD元素数据结构设计

a) LD POU类图



图2-2 POU类图

b) LD元素类结构图

图2-3 图形元素类结构图

在LD逻辑组态中，主要图形元素及其对应的类包括：网络（CLDNetwork）、触点（CLDContact）、跳转（CLDJump）、返回（CLDReturn）、线圈（CLDOutput）、块元件（CLDBox）

有些节点是程序使用的临时节点，在逻辑组态中并不真实存在，这些元素及其对应的类包括：Assign节点（CLDAssign）、OR元件（CLDOr）、And元件（CLDAnd）、Branches元件（CLDBranches）。

c) 文档视图结构与LD的关联



图2-4文档结构视图与LD关系图

d) LD图形元素存储结构

LD图形元素的存储结构是指用户的LD组态中所用到的各种元件在内部的一个存储结构,这结构对用户而言是不可见的。

在用户用梯形图进行组态时，LD编辑器会自动地生成这样的一个结构图。例如当用户添加两个串联的触点时，LD编辑器将会生成一个AND元件（CLDAnd对象），并将两个触点置为AND元件的孩子节点。当用户完成LD的组态程序时，LD编辑器将在内部生成一个完整的树结构，我们将这个树命名为：逻辑存储树结构。可以看出，每个网络节点都对应着一个这样的逻辑存储树结构，一个LD POU是由多个这样的树结构组成的。

这个存储树结构在LD的整个编辑、编译过程中有着重要的作用，包括LD界面的显示、LD语法检查、LD逻辑转换都将遍历该树结构。

这里给出一个具体实例，来说明组态逻辑对应的逻辑存储树结构。

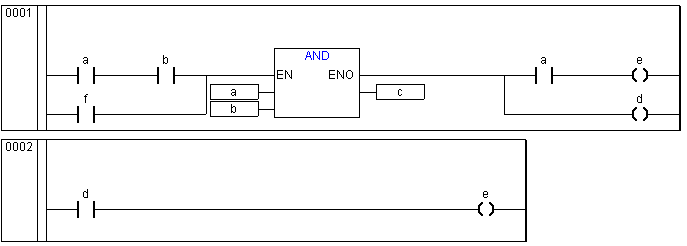


图2-5 LD组态程序图

上面的LD组态程序对应的逻辑存储树结构如下图所示：



图2-6 图2-5LD组态程序图对应的逻辑存储树结构

#### 3. LD图形元素类说明

本节结合DCS\_AT项目，对LD语言的主要图形元素类的成员进行说明，不管是用于代码维护还是今后新需求的开发，都会有指导作用。

（1）CLDElement：是所有元素类的基类，LD图形元素类都继承于该类

在该类中主要的成员有：

int m\_iUpHalf; //元素上部分，以水平电力线划分

int m\_iDownHalf; //元素下部分，以水平电力线划分

CRect m\_virtualRect; //元素实际大小所在矩形

CRect m\_textRect; //元素关联的文本所在矩形

CLineArray m\_aLineArray; //元素包括直线所在数组

CArray<int,int> m\_childrenIDList;//其子结点点的所有ID 号列表

CElementTextInfo m\_typeItem; //元素关联的文本信息

int m\_iParentID; //父结点ID号

int m\_iNetworkID; //记录每一个元素它所在的网络节点的ID号

int m\_iIndex; //当其作为子节点时，是其父节点的第几个节点

int m\_iDeepIndex; //在树中的层次 root = 0

ElementType m\_emTypeID; //类型//Box Input output contact return jump and or等

CString m\_strText; //元素中的具体内容

bool m\_bHeadLogic; //前逻辑布尔值

bool m\_bTailLogic; //后逻辑布尔值

bool m\_bDeleteTag; //删除标志

（2）CLDNetwork：梯形图中的网络节点，主要成员有：

CString m\_strComment; //注释

CString m\_strLabel; //标签

（3）CLDAssign：网络元件的辅助类，主要成员为：

CRect m\_tailRect; //输出区域所在矩形

（4）CLDContact：梯形图中的触点元件，主要成员为：

CBaseDB \*m\_pContactVar; //触点所关联的变量

bool m\_bNegate; //是否置反

CString m\_strComment; //触点注释

CString m\_strAddress; //触点对应变量的地址

CString m\_strAlias; //触点别名

（5）CLDJump：用于跳转元素相关信息的存储及其绘制，主要成员有：

bool m\_bNegate; //是否置反

CString m\_strComment; //注释

CRect m\_rtMoveLeft; //向左移动的小矩形框

CRect m\_rtMoveRight; //向右移动的小矩形框

char m\_chMoveDirection; //移动方向：-1,defaut;0,left;1,right;

（6）CLDOutput：派生于CLDElement，用于线圈相关信息的存储及

其绘制，主要成员有：

CBaseDB \*m\_pOutputVar; //输出元件关联的变量

bool m\_bNegate; //是否置反

bool m\_bImmediatOutput; //是否是立即输出

CString m\_strComment; //注释

CString m\_strAddress; //变量地址

CString m\_strAlias; //变量别名

char m\_chSetReset; //输出元件置位复位

（7）CLDBox:用于算法块信息的存储及其绘制，主要成员有：

CRect m\_commentRect; //注释矩形框

CRect m\_nameRect; //Box名称

CRect m\_boxRect; //高亮也是这个框

CRect m\_aliasRect; //别名框

CList<CLDInPin\*,CLDInPin\*> m\_inputPinList; //输入引脚链表

CList<CLDOutPin\*,CLDOutPin\*> m\_outputPinList; //输出引脚链表

（8）CLDReturn：用于Return元件相关信息的存储及其绘制，主要成员有：

bool m\_bNegate; //是否置反

CRect m\_rtMoveLeft; //左移矩形框

CRect m\_rtMoveRight; //右移矩形框

char m\_chMoveDirection; //移动方向：-1,defaut;0,left;1,right;

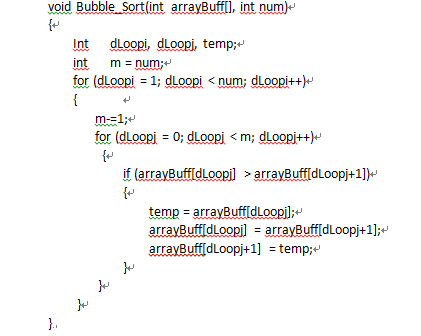
### 三、LD语言的应用场景

LD语言与继电器控制电路图类似，具有直观易懂的优点，很容易被工厂电气人员掌握，特别适用于开关量逻辑控制，LD两侧的垂直公共线称为母线，在分析梯形图的逻辑关系时，为了借用继电器电路图的分析方法，可以想象左右两侧母线之间有一个左正右负的直流电源电压，母线之间有“能流”从左向右流动。

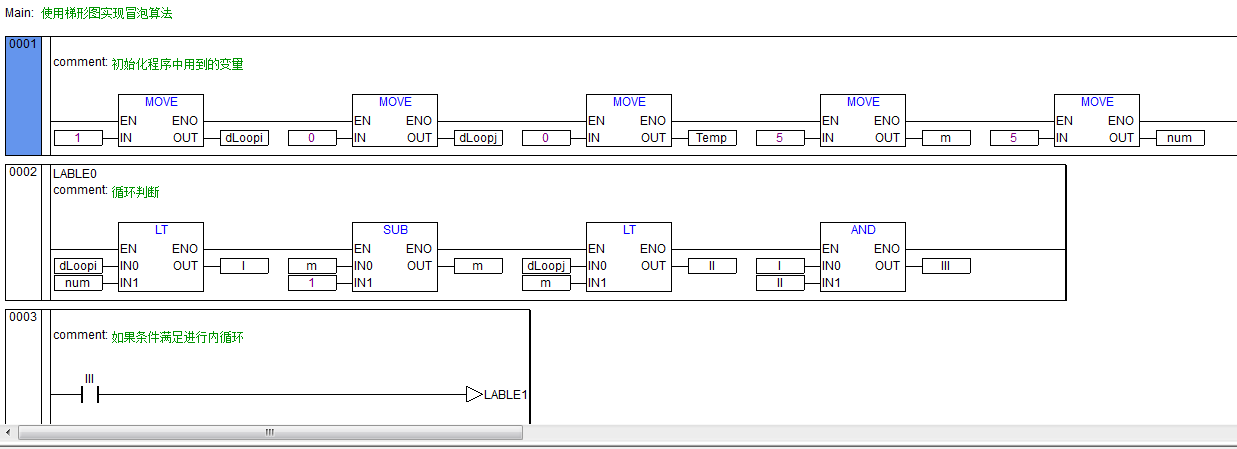
在编制梯形图程序时，如果一个逻辑串很长不便于编辑时，可以将逻辑分成几节（本文中网络），前一段的逻辑运算结果可作为中间输出储存在指定的存储区，该存储区可以当做一个触点出现在其他逻辑串中。

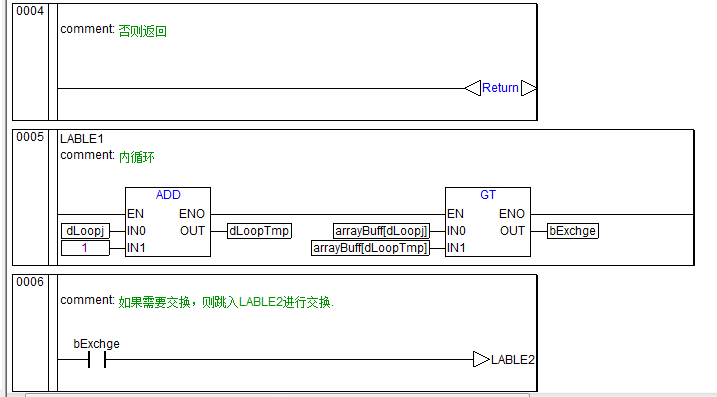
下面通过使用LD语言来实现冒泡排序算法来熟悉LD语言的应用。

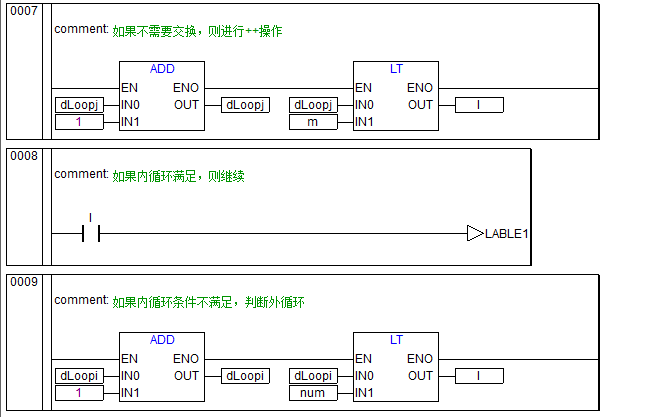
冒泡排序算法的C语言实现：

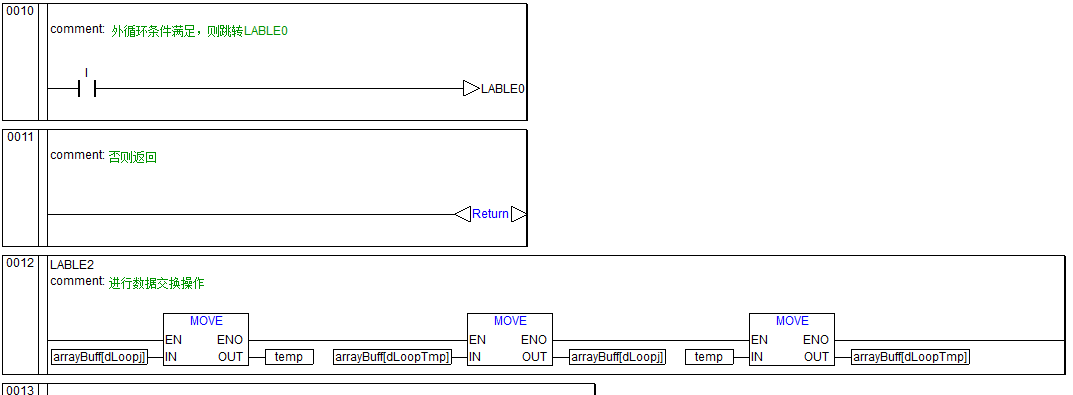


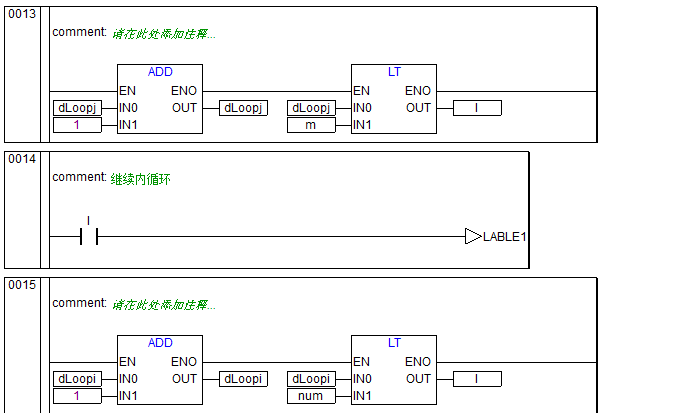
下面用LD梯形图来实现相应的功能：

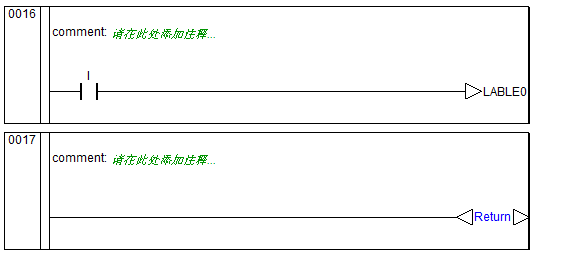
****

****

****

****

****

****

运行结果如下：

仿真前：



仿真后：

