伴随学习与线索推理式自动编程实验

[一、摘要 1](#_Toc7120809)

[二、人类的伴随学习案例 1](#_Toc7120810)

[三、编程模拟伴随学习 1](#_Toc7120811)

[1.元操作 2](#_Toc7120812)

[2.元谓词 2](#_Toc7120813)

[3.因果关系 2](#_Toc7120814)

[4.因果关系相互交叉可以构成语义网络（状态网络）或决策树 4](#_Toc7120815)

[5.面向目标模式 4](#_Toc7120816)

[6.对中间过程的记录 5](#_Toc7120817)

[四、记录负经验（失败经验） 7](#_Toc7120818)

[五、本实验与自动编程的关系 7](#_Toc7120819)

一、摘要

伴随学习的定义：人类在解决一个复杂问题的过程中，会同时解决其中的几个子问题（中间过程）。顺带记录下这些中间过程，为以后的学习提供便利的能力，叫做“伴随学习”能力。伴随学习机制不仅要记录下成功的经验，还要记录下失败的经验（负经验），比如在某情况下尝试某个方法导致失败，则以后遇到此类情况时避免使用此方法，减少尝试的次数，提高学习速度。

二、人类的伴随学习案例

生活的伴随学习例子有很多，例如，我们从瓜渚湖出发，一路上通过看地图、问路，最后到达了大坂湖。中间我正好途径柯桥区政府。我的主要目的是记录下从出发地（瓜渚湖）到目的地（大坂湖）的道路，但是我们会同时顺便记住从瓜渚湖到柯桥区政府的道路——这就是一种伴随学习。

三、编程模拟伴随学习

在基于MFC的Windows窗体程序设计中，常用的字符串类型是CString型，很多操作例如弹出对话框、设置文本框里的文字内容时，接受的参数必须是CString型字符串，而不能是标准的std::string字符串或者原始的字符数组。所以我们只能先将标准字符串/字符数组转化为CString，再传递给函数。

现在我们的主要任务是：

为计算机编写一定的代码，使得它能够将标准字符串转为字符数组，以及将字符数组转为CString。现在给计算机一个标准字符串变量，让它想方设法弹出一个对话框以显示该标准字符串。

如果让程序员来完成这个任务，那么他肯定会写出如下伪代码：

将标准字符串转为字符数组—→将字符数组转为CString—→弹出对话框

我们再将每一步操作前后的变量状态补充进去（用括号括起来），得到路线图如下：

（最开始这是个标准字符串）—→将标准字符串转为字符数组—→（变为字符数组）—→将字符数组转为CString—→（变为CString）—→弹出对话框

按照人类的思维方式，计算机在解决“弹出对话框显示标准字符串”这一主要问题的时候，它也顺便学会了解决如下几个子问题：

1. 将标准字符串转为CString

（最开始这是个标准字符串）—→将标准字符串转为字符数组—→（变为字符数组）—→将字符数组转为CString

1. 弹出对话框显示字符数组

（最开始这是个字符数组）—→将字符数组转为CString—→（变为CString）—→弹出对话框

这就是伴随学习在程序设计中的一个例子。

1.元操作

弹出对话框

标准字符串转为字符数组

字符数组转为CString

整数转为CString

2.元谓词

是整数（int）

是标准字符串（std::string）

是字符数组（char[]）

是CString型字符串

元操作是动词，我这里规定每个元操作函数必须有返回值，返回0（NULL）表示执行失败，返回非零值表示成功。

谓词仅仅用于判定参数的属性，不会对参数对象造成任何影响。如果传入的参数对象具有某属性，则返回true，否则返回false。

所有的元操作、元谓词均接受一个参数。

只有满足一定条件时，才能执行某个元操作函数，例如要在对话框里显示某个字符串，首先这个字符串必须得是CString类型的；再例如要执行函数【标准字符串转化为字符数组】，首先它接受的参数必须得是一个标准型字符串（std::string）。有些函数执行后会对传入的参数变量造成一些影响，比如【将标准字符串转化为字符数组】这个函数，执行以后参数变量的类型就由标准字符串型（std::string）变为字符数组型（char[]）。

3.因果关系

本实验中涉及的“关系”只有两种：

* 1. 一种是【执行的动作】与【造成的影响】之间的联系，前者是一个由动词和参数构成的命令，后者是由谓词和参数构成的命题。

例如【将字符数组转为CString】是执行的动作，【使得参数成为CString型字符串】是造成的影响，

* 1. 另外一种关系是【执行的动作】与【执行此动作所需要满足的前提条件】之间的联系。例如【将字符数组转为CString】，首先你传入的参数变量必须得是字符数组，如果是其他对象，则程序会将该对象强制转型成字符数组导致异常。这里【参数是字符数组】就是执行命令的前提条件，它也是一个由谓词和参数构成的命题。

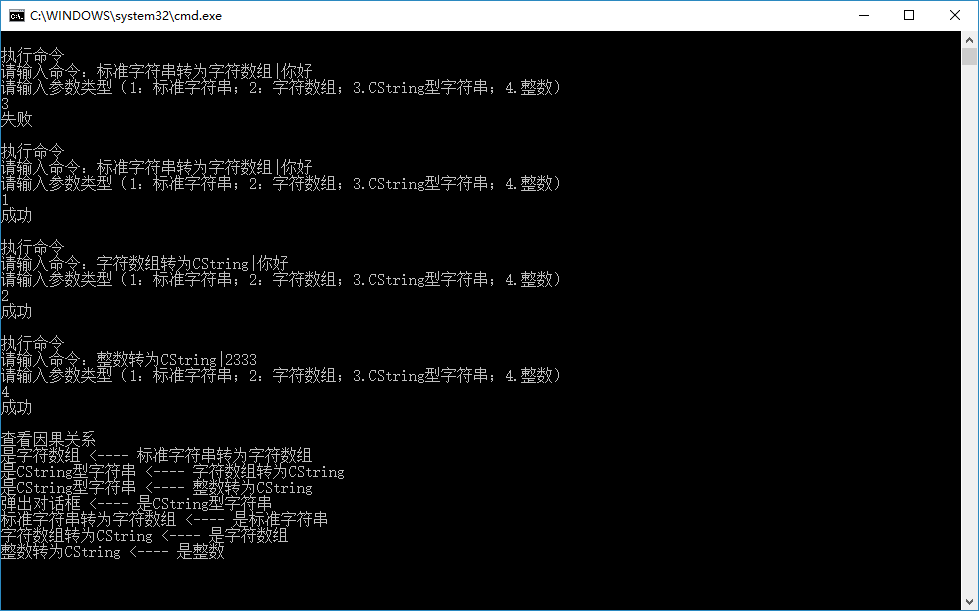
|  |  |  |
| --- | --- | --- |
| **参数需要符合的条件** | **元操作** | **对参数变量造成的影响** |
| 是CString型字符串 | 弹出对话框 | 无 |
| 是标准型字符串 | 标准字符串转为字符数组 | 是字符数组 |
| 是字符数组 | 字符数组转为CString | 是CString型字符串 |
| 是整数 | 整数转为CString | 是CString型字符串 |

上述这些关联，都不是由程序员预先编写好的，需要通过程序自己学习得到。



如上图所示，输入命令，弹出对话框显示CString型字符串“你好”，执行成功。输入“查看因果关系”，可以看到系统记录下了这样一条因果关系：弹出对话框 <---- 是CString型字符串。

意思就是，如果要弹出对话框显示一个字符串，首先要满足的前提条件是“该字符串必须是CString类型的”。



我们输入命令，使用的函数是【标准字符串转为字符数组】，接受的参数是CString型字符串“你好”。很显然，这个命令不能被成功执行，因为参数类型不匹配，要求接受的参数是标准字符串型的，而实际参数却是CString型。因为执行失败，所以程序没有记录下任何的因果关系。

下次输入同样的命令，只不过接受的变量是标准型字符串，这一次命令执行成功了，系统不仅记录下“执行【标准字符串转为字符数组】函数的前提条件是参数必须得是标准型字符串”，还记录下了另外一个因果关系：

是字符数组 <---- 标准字符串转为字符数组

意思就是，执行【标准字符串转为字符数组】这一命令导致参数变量变成了字符数组。

将这两条因果关系联立起来，得到：执行【标准字符串转为字符数组】这一函数，可以使得参数由【标准字符串】转变为【字符数组】。这时，系统便真正理解了执行这一函数的前因与后果。

4.因果关系相互交叉可以构成语义网络（状态网络）或决策树

上一张截图罗列出了本程序中所有的函数所涉及的因果关系。例如【弹出对话框】这一函数只接受【CString型】的变量（前因）；再看“整数转为CString”，它的前提条件是接受参数必须是【整数型】变量（前因），它导致后果是参数变量被转化为【CString型】变量。这样可以看出，状态结点【整数型】与行为结点【弹出对话框】之间存在连通关系，按照如下路径行动，可以在对话框中显示一个整数型变量：

将整数型变量转为CString型 ----> 弹出对话框显示转化后的CString字符串

5.面向目标模式

在常规模式下，执行一个命令，如果失败了，则系统就会直接在屏幕上输出“失败”。

而在面向目标模式下，如果执行命令失败了，系统会思考：如果要使这个命令执行成功，需要满足什么条件？然后它会想方设法使得前提条件成立。

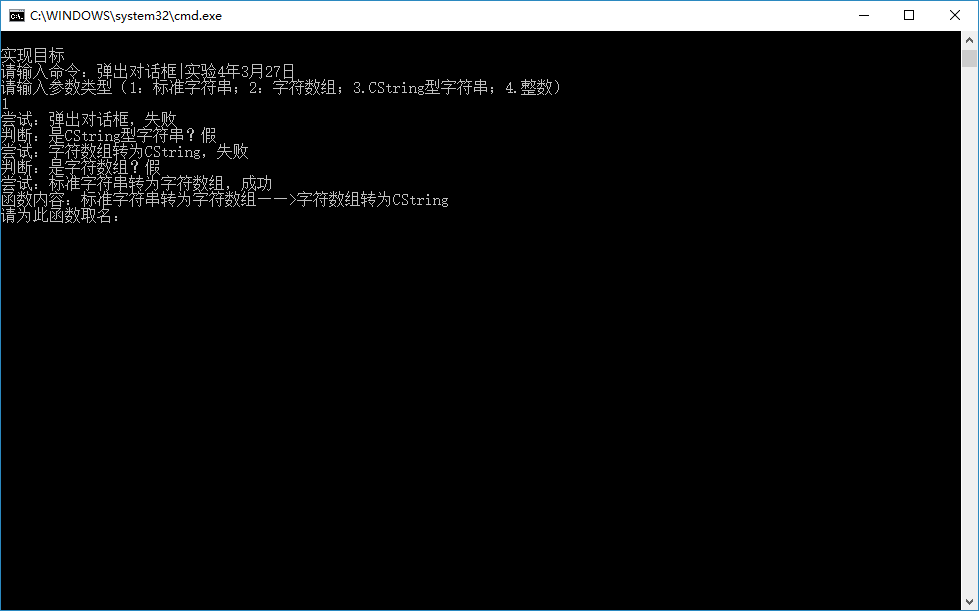
例如给定一个标准型字符串，让系统在对话框里显示这个字符串，由于参数类型不是CString型，执行失败。系统想方设法将参数类型转化为CString型，它了解到调用【将字符数组转为CString】可以达到这一效果，

于是它调用了该函数，结果由于参数不是字符数组，命令执行失败了。

然后系统又想法设法让参数变成字符数组，它发现【将标准字符串转为字符数组】可以满足这一需求，该命令执行成功。

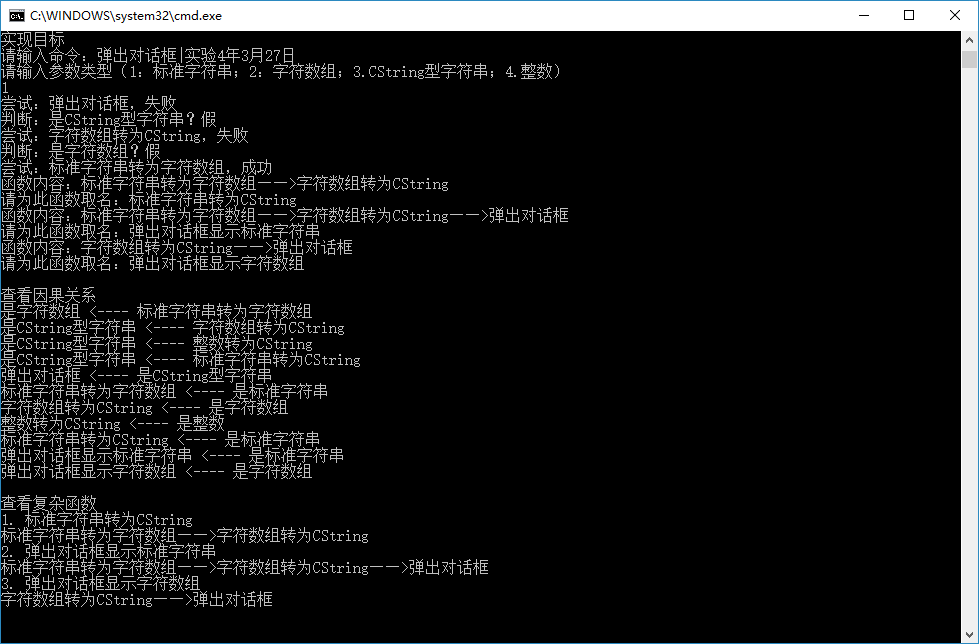
于是系统记录下全过程：将标准字符串转为字符数组—→将字符数组转为CString—→弹出对话框

逐个执行这些命令，最终成功地在对话框里输出了原标准字符串的内容。

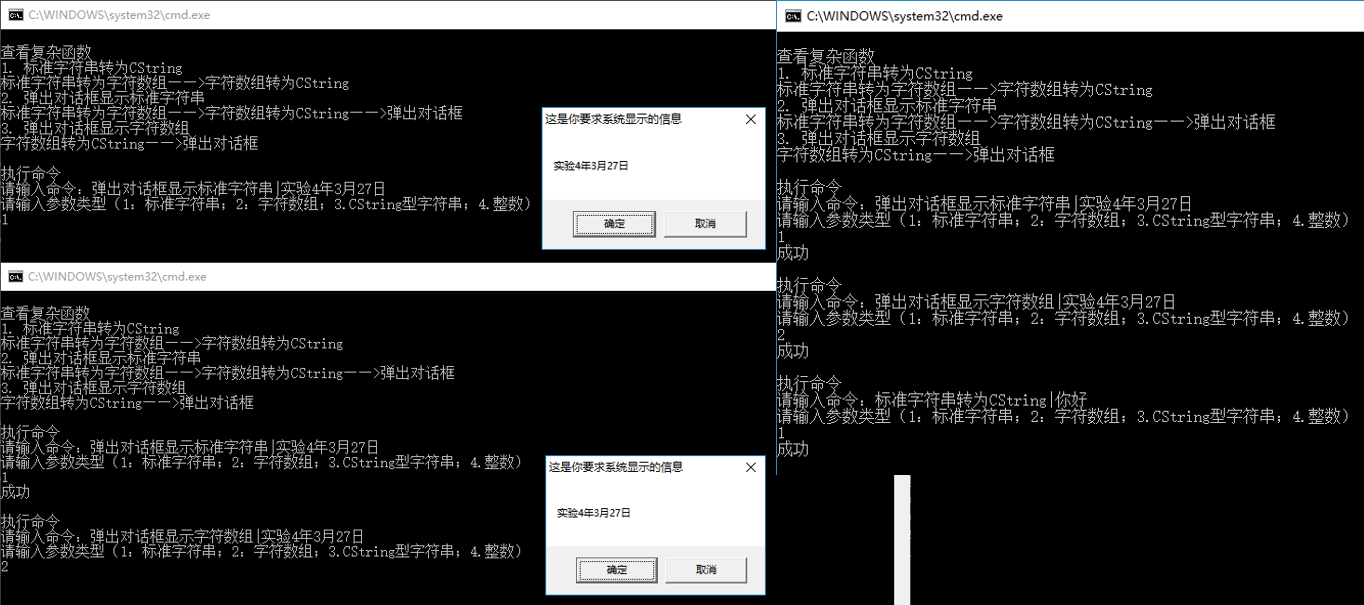


6.对中间过程的记录

下图展示了整个学习的过程，可以看出系统不仅记录下了主任务——如何【弹出对话框显示标准字符串】，还记录下了中间过程——①如何【将标准字符串转为CString】；②如何【弹出对话框显示字符数组】。



学习完毕，下面三张截图展示了运行效果。



四、记录负经验（失败经验）

我在元操作的函数体内写入了异常处理代码，例如【将字符数组转为CString】，如果传入的指针参数如果不是指向字符数组的，则直接返回NULL，不会执行后面的代码。但如果将这一句异常处理代码注释掉呢？很显然整个程序会直接崩溃、终止运行。

我们可以引入一个负经验记录机制，随时随刻地在数据库里面（不要记录在内存里面，因为程序崩溃以后内存的数据就没了）记录系统执行的命令C，以及执行命令之前参数的状态P。如果程序崩溃异常退出，则下次运行程序的时候，读取数据库里的运行数据。并记录下这条负经验——在参数满足状态P的时候执行命令C，会导致程序崩溃。下次再执行命令C的时候，系统会检查参数是否符合状态P，如果满足，则直接返回NULL，避免强制转型导致程序崩溃。经多次训练以后，程序就会习得一套异常处理机制。

五、本实验与自动编程的关系

本实验中，系统通过对给出的几个元操作函数（标准字符串转为字符数组、字符数组转为CString、弹出对话框）进行组合调用，给出了能够【弹出对话框显示标准字符串变量】的指令序列。这就是一个自动编程过程，跟以往不同，这次程序使用的不是暴力穷举的算法，而是用了找线索、递归的方法，根据前因后果来判断自己应该采取的行动，一步一步得出解决问题的方案，这种方法的执行效率更高，更加符合人类的思维方式。