一个关于状态机的问题

我不是计算机专业毕业的，也没有研究过离散数学（听说状态机属于离散数学）。前几天刚好有机会听别人讲这方面的问题，于是针对这个题目研究了一下。记录与此。

一. 问题

请画出车库门控制器的 FSM(finite state machine)。车库门的功能如下：

1. 开启车库门
2. 关闭车库门
3. 车库门进行中（正在开启或关闭中）停止
4. 当车库门是处于第 3 点的状态时，往（停止前）反方向运行

PS：以上这些功能只须使用一个控制键

二. 分析

传统的做法应当是这样的

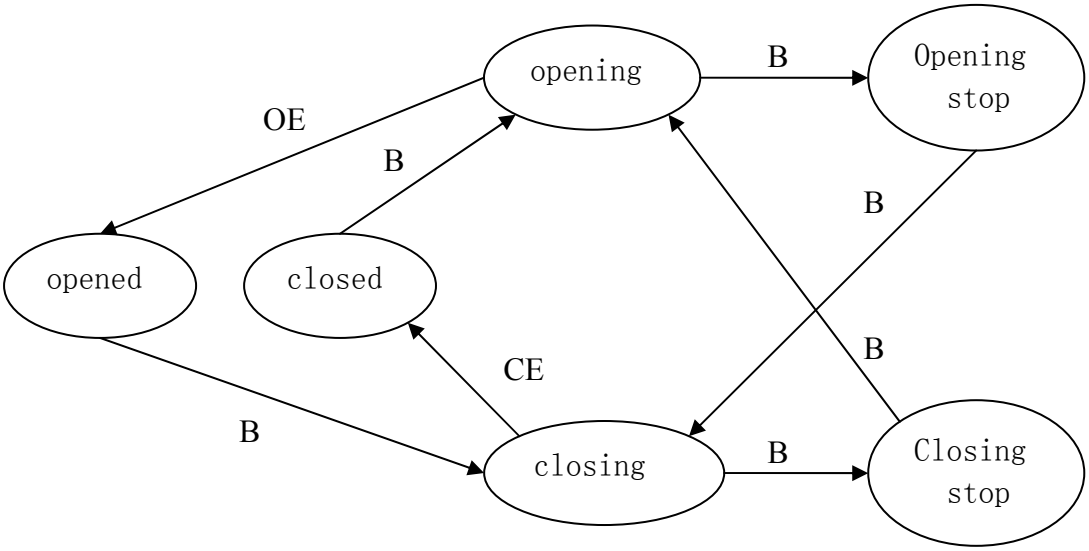
repeat

until 有键按下;

根据键转移状态

这样做很麻烦，结构上不容易控制清晰

或者再高级一些，使用类似于“中断”的方式控制，也存在不容易修改的问题。使用状态机的概念会使得问题清晰（我是刚学）。



上面就是状态图。图中各个状态含义如下：

opened 车库门处于开启状态 closed 车库门处于关闭状态
opening 车库门正在开启 closing 车库门正在关闭
opening stop 车库门在开启时停止 closing stop 车库门在关闭时停止
B 表示按键事件
OE 表示开启动作结束的事件（实际中应当有一个传感器类似东西来传递车门已经打开的信息）
CE 表示开关闭作结束的事件（实际中应当有一个传感器类似东西来传递车门已经关闭的信息）

实际上可以看出，车门的状态转移是由事件(Event)决定的。从一个状态跳转到何种状态则是由当前的状态和事件决定的。从这个角度讲，这种抽象方式我觉得很接近面向对象了（oo 我还不太懂~）

从这出发，可以由一张表来表示这一切：

状态 ｜ 事件	B	OE	CE
Oened (S0)	(S3)		
Closed (S1)	(S2)		
Opening (S2)	(S4)	(S0)	
Closing (S3)	(S5)		(S1)
Opening stop (S4)	(S3)		
Closing stop (S5)	(S2)		

简单起见，对各个状态编号从 S0 到 S5

根据当前状态，根据事件，完全可以决定跳转的下一个状态，这便是状态机的编程。核心就是根据这张表进行状态转移，因此无论增加还是删除状态都可以很轻松的通过修改表格来完成。

三. 具体编程实现

程序的具体说明见最后

```
program StateMac;
uses crt;
const
      N=50;                                          {门的长度，显示用}
type
      DoorType=record
                     current:integer;
                end;
var
    EventTbl       :array[0..5,0..2] of integer;    {状态表}
    Door             :DoorType;                     {记录门打开的程序，这只是
                                                      模拟显示的需要}
    Sender,Event:integer;                           {分别是当前的状态，事件。
                                                      这两个可以放在一个结构体中}

procedure Dispatch;                                 {根据状态表转移，核心}
var
      temp:word;
begin
      temp:=EventTbl[Sender,Event];                 {保存要跳转的状态过程的地址，
                                                      实际上 temp 当作指针来使用的}
   asm
       mov ax,temp
       jmp ax
   end;
end;

procedure DrawDoor(len:integer);                    {根据 len 画门}
var
    i:integer;
begin
         gotoxy(4,20);
         if len=0 then
                write('Opened')                     {门如果是处于打开的状态，
```

```
                                                表示一下}
        else
            write('          ');

        for i:=1 to len-1 do
          write(char(219));                     {画已经关上的门}

        for i:=len to N do
          write(char(176));                     {画没有关上的门}

        if len=N then
            write('Closed')          {门如果是处于关闭的状态，表示一下}
        else
            write('          ');
end;

procedure Closing;
var
    i:integer;
begin
  while true do
    begin
      if KeyPressed then
                begin
                  ReadKey;
                  Sender:=3;
                  Event:=0;
                  asm
                    mov ax,offset Dispatch;          {跳转到处理事件}
                    jmp ax
                  end;
                end;
      if Door.current=N then
                begin
                  Sender:=3;
                  Event:=2;
                  asm
                    mov ax,offset Dispatch;
                    jmp ax
                  end;
                end;
      inc(Door.Current);
      DrawDoor(Door.Current);
      delay(200);
```

```
        end;
end;

procedure Opening;
var
     i:integer;
begin
   while true do
      begin
         if KeyPressed then
                        begin
                           ReadKey;
                           Sender:=2;
                           Event:=0;
                           asm
                              mov ax,offset Dispatch;
                              jmp ax
                           end;
                         end;
         if Door.current=0 then
                        begin
                           Sender:=2;
                           Event:=1;
                           asm
                              mov ax,offset Dispatch;
                              jmp ax
                           end;
                        end;
         dec(Door.Current);
         DrawDoor(Door.Current);
         delay(200);
      end;
end;

procedure Closed;
begin
   DrawDoor(N);
   while true do
      begin
         if KeyPressed then
                        begin
                           ReadKey;
                           Sender:=1;
                           Event:=0;
```

```
                        asm
                           mov ax,offset Dispatch;
                           jmp ax
                        end;
                     end;
      end;
end;

procedure Opened;
begin
   DrawDoor(0);
   while true do
      begin
         if KeyPressed then
                     begin
                        ReadKey;
                        Sender:=0;
                        Event:=0;
                        asm
                           mov ax,offset Dispatch;
                           jmp ax
                        end;
                     end;
      end;
end;

procedure Openingstop;
begin
   while true do
      begin
         if KeyPressed then
                     begin
                        ReadKey;
                        Sender:=4;
                        Event:=0;
                        asm
                           mov ax,offset Dispatch;
                           jmp ax
                        end;
                     end;
      end;
end;

procedure Closingstop;
```

```
begin
   while true do
      begin
         if KeyPressed then
                        begin
                           ReadKey;
                           Sender:=5;
                           Event:=0;
                           asm
                              mov ax,offset Dispatch;
                              jmp ax
                           end;
                        end;
      end;
end;


begin
   clrscr;
   Door.current:=0;

{转移表赋初始值}
   EventTbl[0,0]:=ofs(Closing);
   EventTbl[1,0]:=ofs(Opening);
   EventTbl[2,0]:=ofs(Openingstop);
   EventTbl[2,1]:=ofs(Opened);
   EventTbl[3,0]:=ofs(Closingstop);
   EventTbl[3,2]:=ofs(Closed);
   EventTbl[4,0]:=ofs(Closing);
   EventTbl[5,0]:=ofs(Opening);

   asm
     mov ax,offset Opened;
     jmp ax
   end;
end.
```

这个程序定义了很多过程，仔细看会发现调用他们的方式都很特别，都不是普通的方式，而是将地址放入寄存器然后 jmp 过去的。这是因为如果按照普通方式，调用过程中会有压栈的动作，执行起来最后会溢出或者根本无法正常退出。我写过一点面向对象的 pascal 程序，当时非常奇怪，因为都是发 event 过

去处理，按照一般的过程的调用，堆栈一直变大的~现在想起来，那个程序在编译中也一定是像这样处理的。面向过程和面向对象还是有很大差别地~

程序很不完善，至少没有写退出的过程；并且数据结构不是非常清晰的，前面的状态应该用枚举类型~

写在这里算作“存此立照”吧。

Thursday, November 04, 2004

Z&Z