C语言实现有限状态机

有限状态机是一种数学概念，运用到程序中，可用于有限数量的状态的变化，每个子程序进行一些处理并选择下一种状态。

基本的实现思路是用一张表保存所有可能的状态，并列出进入每个状态时可能执行的所有动作，其中后一个动作是计算下一个应该进入的状态。运行状态是从初始状态开始，不停的在各个状态之间转换，直到结束状态。

FSM的实现方式：

1）switch/case或者if/else

这无意是直观的方式，使用一堆条件判断，会编程的人都可以做到，对简单小巧的状态机来说合适，但是毫无疑问，这样的方式比较原始，对庞大的状态机难以维护。

2）状态表

维护一个二维状态表，横坐标表示当前状态，纵坐标表示输入，表中一个元素存储下一个状态和对应的操作。这一招易于维护，但是运行时间和存储空间的代价较大。

3）使用State Pattern

使用State Pattern使得代码的维护比switch/case方式稍好，性能上也不会有很多的影响，但是也不是100%完美。不过Robert C. Martin做了两个自动产生FSM代码的工具，for java和for C++各一个，在http://www.objectmentor.com/resources/index上有免费下载，这个工具的输入是纯文本的状态机描述，自动产生符合State Pattern的代码，这样developer的工作只需要维护状态机的文本描述，每必要冒引入bug的风险去维护code。

4）使用宏定义描述状态机

一般来说，C++编程中应该避免使用#define，但是这主要是因为如果用宏来定义函数的话，很容易产生这样那样的问题，但是巧妙的使用，还是能够产生奇妙的效果。MFC是使用宏定义来实现大的架构的。

在实现FSM的时候，可以把一些繁琐无比的if/else还有花括号的组合放在宏中，这样，在代码中可以3）中状态机描述文本一样写，通过编译器的预编译处理产生1）一样的效果，我见过产生C代码的宏，如果要产生C++代码，已软MFC可以，那么理论上也是可行的。

密码锁的例子

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef enum{
 STATE0 = 0，
 STATE1，
 STATE2，
 STATE3，
 STATE4，
}STATE;


int main()

{
 char ch;
 STATE current_state = STATE0;
 while(1){
  printf("In put password:");
  while((ch = getchar()) != '
')
  {
   if((ch < '0') || (ch > '9'))
   {
    printf("Input num，ok?/n");
    break;
   }
   switch(current_state){
   case STATE0:
    if(ch == '2')   current_state = STATE1;
    break;
```

```
   case STATE1:
    if(ch == '4')   current_state = STATE2;
    break;
   case STATE2:
    if(ch == '7')   current_state = STATE3;
    break;
   case STATE3:
    if(ch == '9')   current_state = STATE4;
    break;
   default:
    current_state = STATE0;
    break;
   }
  }

  if(current_state == STATE4){
   printf("Correct ,  lock is open!
");
   current_state =   STATE0;

  }else
  {
   printf("Wrong ,  locked!
");
   current_state =   STATE0;

  }
  break;
 }
 return 0;

}
```