指出这个问题是为了说明使用信号量时要非常小心。一处很小的错误将导致很大的麻烦。这就像用汇编语言编程一样，甚至更糟，因为这里出现的错误都是竞争条件、死锁以及其他一些不可预测和不可再现的行为。

为了更易于编写正确的程序，Brinch Hansen（1973）和Hoare（1974）提出了一种高级同步原语，称为管程（monitor）。在下面的介绍中我们会发现，他们两人提出的方案略有不同。一个管程是一个由过程、变量及数据结构等组成的一个集合，它们组成一个特殊的模块或软件包。进程可在任何需要的时候调用管程中的过程，但它们不能在管程之外声明的过程中直接访问管程内的数据结构。图2-33展示了用一种抽象的、类Pascal语言描述的管程。这里不能使用C语言，因为管程是语言概念而C语言并不支持它。

```
monitor example
        integer i;
        condition c;

        procedure producer( );
        .
        .
        .
        end;

        procedure consumer( );
        .
        .
        .
        end;
end monitor;
```

图2-33 管程

管程有一个很重要的特性，即任一时刻管程中只能有一个活跃进程，这一特性使管程能有效地完成互斥。管程是编程语言的组成部分，编译器知道它们的特殊性，因此可以采用与其他过程调用不同的方法来处理对管程的调用。典型的处理方法是，当一个进程调用管程过程时，该过程中的前几条指令将检查在管程中是否有其他的活跃进程。如果有，调用进程将被挂起，直到另一个进程离开管程将其唤醒。如果没有活跃进程在使用管程，则该调用进程可以进入。

进入管程时的互斥由编译器负责，但通常的做法是用一个互斥量或二元信号量。因为是由编译器而非程序员来安排互斥，所以出错的可能性要小得多。在任一时刻，写管程的人无须关心编译器是如何实现互斥的。他只需知道将所有的临界区转换成管程过程即可，决不会有两个进程同时执行临界区中的代码。

尽管如我们上边所看到的，管程提供了一种实现互斥的简便途径，但这还不够。我们还需要一种办法使得进程在无法继续运行时被阻塞。在生产者-消费者问题中，很容易将针对缓冲区满和缓冲区空的测试放到管程过程中，但是生产者在发现缓冲区满的时候如何阻塞呢？

解决的方法是引入条件变量（condition variables）以及相关的两个操作：wait和signal。当一个管程过程发现它无法继续运行时（例如，生产者发现缓冲区满），它会在某个条件变量上（如full）执行wait操作。该操作导致调用进程自身阻塞，并且还将另一个以前等在管程之外的进程调入管程。在前面介绍pthread时我们已经看到条件变量及其操作了。

另一个进程，比如消费者，可以唤醒正在睡眠的伙伴进程，这可以通过对其伙伴正在等待的一个条件变量执行signal完成。为了避免管程中同时有两个活跃进程，我们需要一条规则来通知在signal之后该怎么办。Hoare建议让新唤醒的进程运行，而挂起另一个进程。Brinch Hansen则建议执行signal的进程必须立即退出管程，即

```
monitor ProducerConsumer
    condition full, empty;
    integer count;

    procedure insert(item: integer);
    begin
        if count = N then wait(full);
        insert_item(item);
        count := count + 1;
        if count = 1 then signal(empty)
    end;

    function remove: integer;
    begin
        if count = 0 then wait(empty);
        remove = remove_item;
        count := count - 1;
        if count = N - 1 then signal(full)
    end;

    count := 0;
end monitor;

procedure producer;
begin
    while true do
    begin
        item = produce_item;
        ProducerConsumer.insert(item)
    end
end;

procedure consumer;
begin
    while true do
    begin
        item = ProducerConsumer.remove;
        consume_item(item)
    end
end;
```

图2-34 用管程实现的生产者-消费者问题的解法框架。一次只能有一个管程过程活跃。其中的缓冲区有N个槽