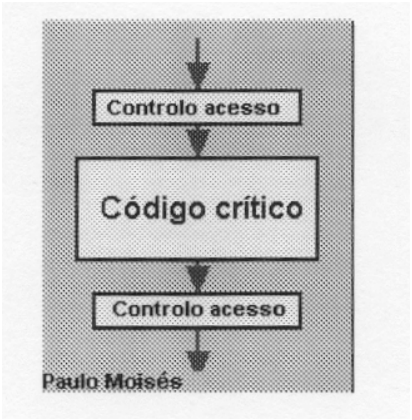
Regiões críticas para exclusão mútua

Semáforos implicam a criação de protocolos de entrada e saída separados:

Esta separação de protocolos encoraja a distribuição das secções criticas ao longo do programa.

Riscos de mau posicionamento ou esquecimento das operações WAIT e SIGNAL, da inicialização do semáforo.



A SOLUÇÃO MAIS ADEQUADA É A DE INTEGRAR A SECÇÃO CRÍTICA NUMA ÚNICA CONSTRUÇÃO:

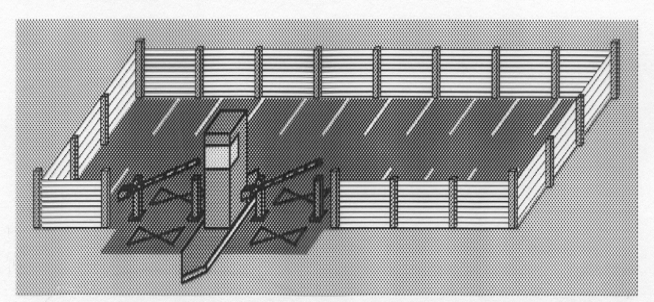

```
N : shared INTEGER := 1;

...

task INCREMENT;

task body INCREMENT is
begin
  region N is
    N := N + 1;
    PUT (N);
  end region;
end INCREMENT;
```

Exemplo com sintaxe semelhante ao ADA, mas não é ADA, porque esta linguagem não dispõe deste mecanismo

- **Cada região é identificada com uma variável identificada como *shared***
- **Qualquer tentativa de acesso a uma variável shared sem ser através de uma região crítica origina um erro pelo compilador**

• A variável shared tem associada uma fila de espera onde são colocados os processos que tentam aceder à variável quando ela está "ocupada" por outro processo.

• Quando um processo termina o seu acesso à zona critica, a fila é verificada e o processo suspenso seguinte é activado.



```
...
loop
    region SPACES is
            GOT_IN := SPACES > 0;
            if GOT_IN then
                    SPACES := SPACES - 1;
            end if;
    end region;
    exit when GOT_IN;
end loop;
...
```

```
...

  region SPACES when SPACES > 0 is
    SPACES := SPACES - 1;
  end region;

...
```

As regiões podem ser utilizadas com guardas.

OS PROBLEMAS DAS REGIÕES CRÍTICAS

As regiões críticas permitem resolver problemas presentes com variáveis partilhadas e Semáforos, tais como:

***A combinação apertada dos protocolos de entrada e de Saída*, uma vez que a natureza fechada das regiões críticas garante que o compilador fecha correctamente as regiões críticas.**

As declarações que são cobertas pela região críticas são claramente identificáveis pelo programador.

***A falta de significado semântico*, é eliminada pelas regiões críticas, uma vez que cada região crítica é associada a uma e uma só uma variável.**

No entanto...

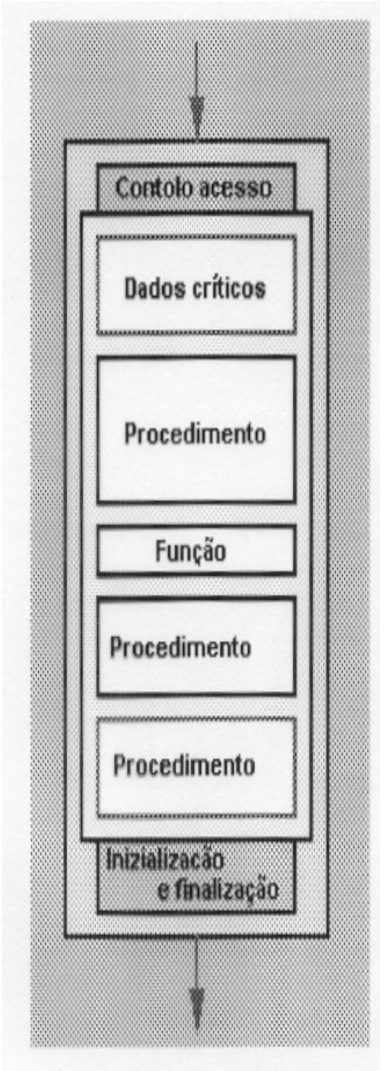
```
region X is
   region Y is
      X := Y;
   end region;
end region;
```

Tarefa A

```
region Y is
   region X is
      X := Y;
   end region;
end region;
```

Tarefa B

Monitores



- **Esta construção pode resolver os problemas das regiões críticas**
- **Um monitor é uma construção fechada semelhante a um pacote**
- **Os dados são declarados como privados no interior do monitor**
- **São facultados vários procedimentos e funções que executam operações sobre os dados protegidos**
- **O acesso aos dados, fora do monitor, é efectuado pelos procedimentos e funções facultados.**
- **Os monitores, contrariamente aos pacotes, têm um mecanismo de controlo de acesso**
- **Existe uma fila de espera associada ao monitor onde são colocados processos que tentem aceder ao monitor quando ele está indisponível.**
- **Quando um processo termina e abandona o monitor, o processo suspenso no princípio da fila de espera irá ser activado.**
- **TODAS AS OPERAÇÕES DENTRO DO MONITOR SÃO ATÓMICAS, OU SEJA OS MONITORES SÃO ESTRUTURAS CUJAS OPERAÇÕES SE EXECUTAM EM EXCLUSÃO MÚTUA.**

- *Um processo que espera uma condição liberta o monitor*

A LINGUAGEM ADA NÃO DISPÕE DE MONITORES.

O CÓDIGO EXEMPLO QUE SE SEGUE USA UM PSEUDO-CÓDIGO ADA (COM SINTAXE SEMELHANTE):

```
monitor ACCESS_CONTROL is

   procedure ADD (AMOUNT : in INTEGER);
   procedure SUBTRACT (AMOUNT : in INTEGER);;
   function RESULT return INTEGER;

private

   VALUE : INTEGER := 0;

end ACCESS_CONTROL;


monitor body ACCESS_CONTROL is

   procedure ADD (AMOUNT : in INTEGER) is
   begin
      VALUE := VALUE + AMOUNT;
   end ADD;

   procedure SUBTRACT (AMOUNT : in INTEGER) is
   begin
      VALUE := VALUE - AMOUNT;
   end SUBTRACT;

   function RESULT return INTEGER is
   begin
      return VALUE;
   end RESULT;

end ACCESS_CONTROL;
```

Uma tarefa que pretenda incrementar os dados dentro do monitor deve chamadar a operação do monitor ADD da seguinte forma:

```
Access_Control.ADD(1);
```

Time	Tarefa Increment	Tarefa Decrement
1	ACCESS_CONTROL.ADD(1);	....
2		ACCESS_CONTROL.SUBTRACT(1);
3	(enter ACCESS_CONTROL)	
4		(suspended on entrance queue)
5	VALUE := VALUE + 1;	
6	(leave ACCESS_CONTROL)	
7		(enter ACCESS_CONTROL)
8	(other processing)	
9		VALUE := VALUE - 1;
10	(other processing)	
11	....	(leave ACCESS_CONTROL)

Linguagens que usam monitores:

Pascal Plus

Concurrent Pascal

Concurrent Euclid