


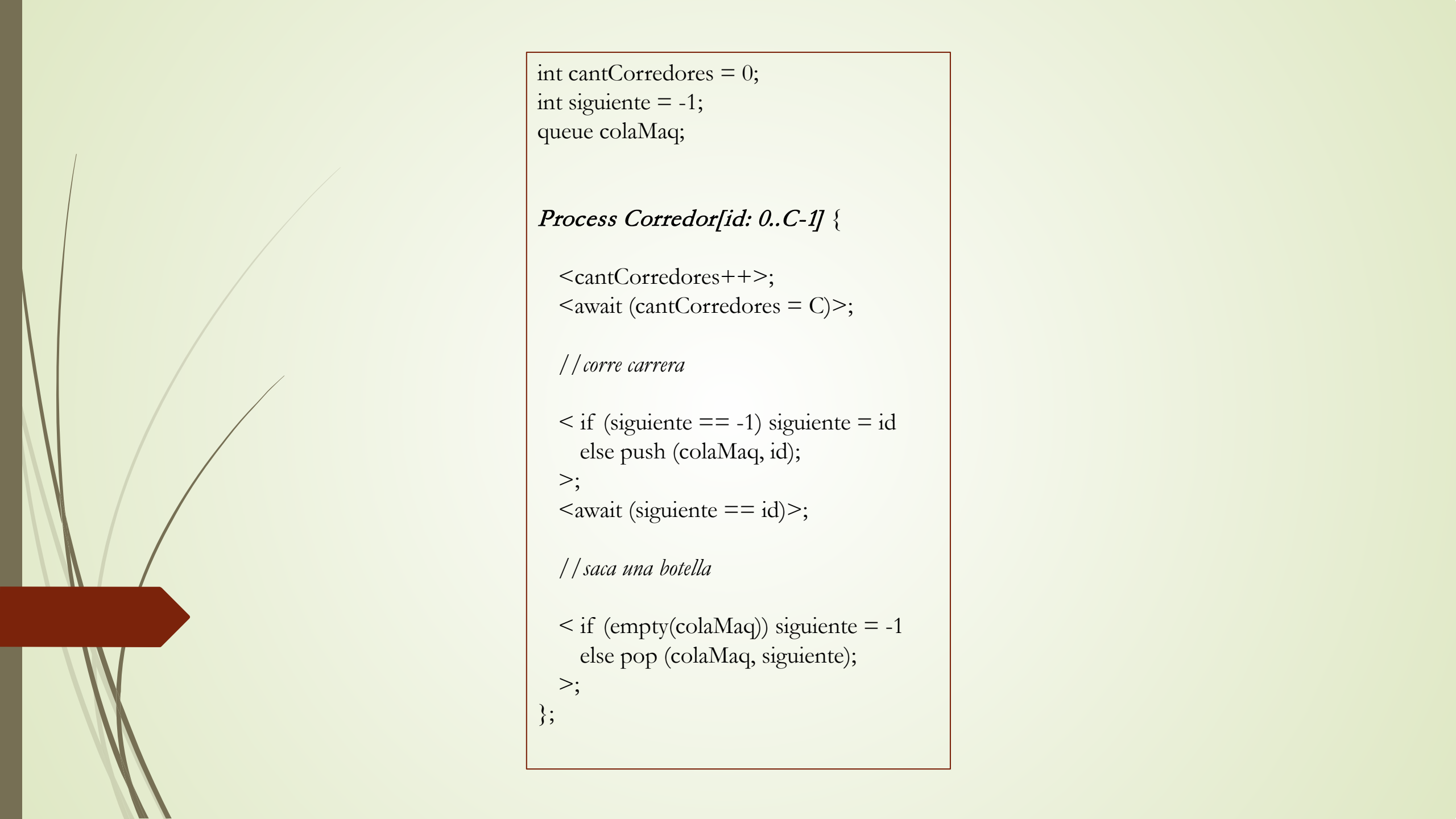


# Posibles soluciones a los ejercicios del parcial

Estas son posibles soluciones, no significa que sean la única forma, traté de hacerlas simples y siguiendo las ideas que hemos visto en las teorías y/o explicaciones prácticas para no confundir.



Resolver con **SENTENCIAS AWAIT** (<> y/o <await B; S>) el siguiente problema. Se debe simular una maratón con **C corredores** donde en la llegada hay UNA máquinas expendedoras de botella de agua. Cuando los C corredores han llegado al inicio comienza la carrera. Cuando un corredor termina la carrera se dirige a la máquina expendedora, espera su turno (respetando el orden de llegada), saca una botella y se retira. **Nota:** maximizar la concurrencia.



```
int cantCorredores = 0;  
int siguiente = -1;  
queue colaMaq;
```

```
Process Corredor[id: 0..C-1] {
```


```
    <cantCorredores++>;  
    <await (cantCorredores = C)>;
```

```
    // corre carrera
```

```
    < if (siguiente == -1) siguiente = id  
      else push (colaMaq, id);  
    >;  
    <await (siguiente == id)>;
```

```
    // saca una botella
```

```
    < if (empty(colaMaq)) siguiente = -1  
      else pop (colaMaq, siguiente);  
    >;  
};
```



Resolver con **SEMÁFOROS** el siguiente problema. Una empresa de turismo posee **4 combis** con capacidad para 25 personas cada una y **UN vendedor** que vende los pasajes a los clientes de acuerdo al orden de llegada. Hay  **$C$  clientes** que al llegar intentan comprar un pasaje para una combi en particular (el cliente conoce este dato); si aún hay lugar en la combi seleccionada se le da el pasaje y se dirige hacia la combi; en caso contrario se retira. Cada combi espera a que suban los 25 pasajeros, luego realiza el viaje, y cuando llega al destino deja bajar a todos los pasajeros. **Nota:** maximizar la concurrencia; suponga que para cada combi al menos 25 clientes intentarán comprar pasaje.

```

sem mutexCola = 1;
sem hayCliente = 0;
sem subioCliente[4] = ([4] 0);
sem esperarFinViaje[4] = ([4] 0);
sem atendido[C] = ([C] 0);

bool pasaje[C];
queue cola;

```

```

Process Cliente [id: 0..C-1] {
    int numC = //combi seleccionada;

    P(mutexCola);
    push(cola, (id, numC));
    V(mutexCola);
    V(hayCliente);

    P(atendido[id]);
    if (pasaje[id]) {
        V(subioCliente[numC]);
        P(esperarFinViaje[numC]);
    };
};

```

```

Process Combi [id: 0..3] {
    int i;

    for (i = 0; i < 25; i++) P(subioCliente[id]);
    // Realiza el viaje
    for (i = 0; i < 25; i++) V(esperarFinViaje[id]);
};


```

```

Process Vendedor {
    int cantV[4] = ([4] 0);
    int i;

    for (i=0; i<C; i++) {
        P(hayCliente);
        P(mutexCola);
        pop(cola, (idCli, numC));
        V(mutexCola);
        if (cantV[numC] < 25) {
            cantV[numC] ++;
            pasaje[idCli] = true;
        }
        else {
            pasaje[idCli] = false;
        };
        V(atendido[idCli]);
    };
};

```



Resolver con **MONITORES** la siguiente situación. Se debe simular un juego en el que participan **30 jugadores** que forman 5 grupos de 6 personas. Al llegar cada jugador debe buscar las instrucciones y el grupo al que pertenece en un cofre de cemento privado para cada uno; para esto deben usar un único martillo gigante de a uno a la vez y de acuerdo al orden de llegada. Luego se debe juntar con el resto de los integrantes de su grupo y los 6 juntos realizan las acciones que indican sus instrucciones. Cuando un grupo termina su juego le avisa a un Coordinador que le indica en qué orden término el grupo. **Nota:** maximizar la concurrencia; suponer que existe una función *Jugar()* que simula que los 6 integrantes de un grupo están jugando juntos; suponga que existe una función *Romper(grupo)* que simula cuando un jugador está rompiendo su cofre con el martillo y le retorna el grupo al que pertenece.



```

Process Jugador[id: 0..29] {
    int numG, respuesta;

    AdminMartillo.Solicitar();
    Romper(numG);
    AdminMartillo.Dejar();

    Grupo[numG].Juntarse(respuesta);
};

```

### **Monitor AdminMartillo {**

```

    bool libre = true;
    int cant = 0;
    cond espera;

```

#### **Procedure Solicitar {**

```

    if (libre) libre = false
    else {
        cant++;
        wait (espera);
    };

```

```

};

```

#### **Procedure Dejar {**

```

    if (cant == 0) libre = true
    else {
        cant--;
        signal(espera);
    };

```

```

};

```

```

}

```

### **Monitor Coordinador {**

```

    int actual = 1;

```

#### **Procedure Resultado (int OUT posFinal) {**

```

    posFinal = actual;
    actual++;

```

```

};

```

```

}

```

### **Monitor Grupo[id: 0..5] {**

```

    int cantidad = 0, posicion = 0;
    cond espera;

```

#### **Procedure Juntarse (int OUT pos) {**

```

    cantidad++;
    if (cantidad < 6) wait (espera)
    else {
        Jugar();
        Coordinador.Resultado(posicion);
        signal_all (espera);
    };

```

```

    pos = posicion;

```

```

};

```

```

}

```