Mohamed Boudiaf University – M'sila
Department of Computer Science
3rd year, Computer Science - SI

First Name:	**Last Name:**	**TEST II / G 1**

A fast-food restaurant can accommodate a limited number of customers. This limit is represented by the number **T** of tables available in the restaurant.

- A customer who arrives washes their hands before sitting at a table; there are only **two (02) washrooms**, each for single use.
- Then the customer waits in a waiting room if no table is available.
- When a table becomes available, the customer sits down and eats.
- As soon as the customer leaves the restaurant, the table is freed for another customer.

Customers can be considered as parallel processes, and the tables and the washrooms as shared resources. Complete the **Monitor-based synchronization algorithm** below for this problem.

```
// specify if the task is in or outside the monitor
Process Customer-i
begin
   FF.enterWashroom();
    washing hands ();
   FF.enterfastfood ();
    Eating ();
   FF. exitfastfood();

end;
```

```
// main program
Program FAST-FOOD;

Begin


Parbegin


Custumer1; Customer2 ;……..;customer-i;



Parend


end.
```

```
//monitor declaration
Monitor FF
VAR

   freeTables: integer;
   freeWashrooms: integer;
   waitForTable: condition;
   waitForWashroom : condition;


 const T=…..;
```

```
// monitor initialization
Begin


   freeTables := T;
   freeWashrooms := 2



End;
```

```
procedure enterWashroom()
begin
    if freeWashrooms == 0 then
        waitForWashroom.wait;
    freeWashrooms = freeWashrooms – 1;
end;
```

```
procedure enterfastfood()
begin
    freeWashrooms = freeWashrooms + 1;
    waitForWashroom.signal;
    if freeTables == 0 then  waitForTable.wait;
    freeTables = freeTables – 1;
end;
```

```
procedure exitfastfood()
begin
    freeTables = freeTables + 1;
    waitForTable.signal;
end;
```

Mohamed Boudiaf University – M'sila
Department of Computer Science
3rd year, Computer Science - SI

SOLUTION	**TEST II / G 2**

A car service center has a **limited number of parking spots**. This limit is represented by the number **P** of parking spaces available in the center. There are two types of cars: **regular** and **large**. Regular cars need one spot; however, large cars occupy **two parking spots** instead of one (They must wait until **two spots** are available):

- A car that arrives waits in a **waiting lane** if no parking spot is available.
- Before parking, each car must go through a **car wash**; there are only **three (03) washing stations**, each for **single use**.
- Once the car is washed, it parks in an available spot (or two spots for large cars).
- As soon as the car leaves the parking spot, it is freed for another car.

Cars can be considered as parallel processes, and the parking spots and washing stations as shared resources. Complete the **Monitor-based synchronization algorithm** below for this problem.

```
// specify the tasks in the monitor
Process Car-i (size) // size=1 for regular, 2 for large
begin
        P.enterParking(size);
        wash-car();
        P.leaveWash();
        park-car ();
        P.releaseParking(size);
end;
```

```
// main program
Program PARKINK;
Begin
parbegin
car1 (size); car2(size); …..; car-i(size);
parend
end.
```

```
//monitor declaration
Monitor P
VAR
   freeSpots: integer;
   freeWashStations: integer;
   waitForWash: condition;
   waitForParking: condition;
const P=…..;
```

```
// monitor initialization
Begin
freeSpots = P;
freeWashStations = 3;
End;
```

```
procedure enterParking (size: interger)
begin
    while freeSpots < size then waitForParking.wait;
    freeSpots = freeSpots – size;
    if freeWashStations == 0 then waitForWash.wait;
    freeWashStations = freeWashStations – 1;
end;
```

```
procedure leaveWash()
 begin
    freeWashStations = freeWashStations + 1;
    waitForWash.signal;
end;
```

```
procedure releaseParking (size: integer)
begin
    freeSpots = freeSpots + size;
    waitForParking.signal;
end;
```

First Name:	**Last Name:**	**TEST II / G 3**

A store can accommodate a limited number of customers. This limit is represented by the number **N** of shopping carts available at the store entrance.

- A customer who arrives waits if no shopping cart is available.
- When a customer gets a shopping cart, they enter the store to do their shopping.
- Then they go to the checkout to pay for their purchases in mutual exclusion, i.e., there is only one checkout.
- As soon as they finish, they release their shopping cart when leaving the store.

Customers can be considered as parallel processes, and the shopping carts and the checkout as shared resources. Complete the **Monitor-based** synchronization algorithm below of this problem:

```
// specify the tasks in the monitor
Process Customer-i
begin
   Store.acquireCart();
   shopping ();
   Store.useCheckout();
    Paying-purchases ();
   Store.leavestore();
end ;
```

```
// main program
Program SHOPPING STORE;

Begin
parbegin

Customer1; customer2; …..; customeri;

parend

end.
```

```
//monitor declaration
Monitor Store
VAR

      carts: integer;
      noCart : condition
      checkoutBusy: condition
      checkoutFree : boolean;

const N =…..;
```

```
// monitor initialization
Begin

   carts:= N;

   checkoutfree:= true;

End;
```

```
procedure acquireCart()
begin
    if carts == 0 then noCart.wait;
    carts = carts – 1;
 end ;


procedure useCheckout()
begin
    if checkoutFree == false then
                checkoutBusy.wait;
    checkoutFree = false;
end;
```

```
procedure leavestore()
begin
    checkoutFree = true;
    checkoutBusy.signal;
    carts = carts + 1;
    noCart.signal;
end;
```