Laundromat

- A Laundromat has 30 washing machines and 15 dryers. Assuming that all customers use one washing machine to wash their clothes then one dryer to dry them, add the necessary semaphore calls to the following program segment.

Laundromat

```
semaphore _____ = _____;
semaphore _____ = _____;
customer (int who) {
        _____;
        wash_clothes();
        _____;
        _____;
        dry_clothes();
                _____; }
```

Laundromat

```
semaphore _washers_____ = __30__;
semaphore _dryers_____ = __15__;
customer (int who) {
        _P(&washer);_____;
        wash_clothes();
        _V(&washer);_____;
        _P(&dryer);_____;
        dry_clothes();
        V(&dryer)          ; }
```

The ice-cream parlor

- An ice-cream parlor has two employees selling ice cream and six seats for its customers. Each employee can only serve one customer at a time and each seat can only accommodate one customer at a time. Add the required semaphores to the following program skeleton to guarantee that customers will never have to wait for a chair with a melting ice-cream in their hand.

The ice-cream parlor (cont'd)

```
semaphore _____ = _____;
semaphore _____ = _____;
customer (int who) {
        _____
        order_ice_cream();
        _____
        eat_it();
        _____
} // customer
```

Sketching a solution

- Two resources are shared by all customers
  - Six seats
  - Two employees
- Questions to ask are
  - When should we request a resource?
  - In which order? (***very important***)
  - When should we release it?

Solution

```
semaphore _seats_____ = __6___;
semaphore _employees_____ = __2___;
customer (int who) {
    _____
    order_ice_cream();
    _____
    eat_it();
    _____
} // customer
```

Solution (cont'd)

```
semaphore _seats_____ = __6___;
semaphore _employees_____ = __2___;
customer (int who) {
    P(&seats); P(&employees);_____
    order_ice_cream();
    _____;
    eat_it();
    _____;
} // customer
```

Get seat first

Solution (cont'd)

```
semaphore _seats_____ = __6___;
semaphore _employees_____ = __2___;
customer (int who) {
    P(&seats); P(&employees);_____
    order_ice_cream();
    V(&employees);_____
    eat_it();
    _____
} // customer
```

What is missing?

Solution (cont'd)

```
semaphore _seats_____ = __6___;
semaphore _employees_____ = __2___;
customer (int who) {
    P(&seats); P(&employees);_____
    order_ice_cream();
    V(&employees);_____
    eat_it();
    V(&seat);_____
} // customer
```

The pizza oven

A pizza oven can contain nine pizzas but the oven narrow opening allows only one cook at a time to either put a pizza in the oven or to take one out. Given that there will be more than one cook preparing pizzas at any given time, complete the missing lines in the following C procedure.

The pizza oven (cont'd)

```
semaphore oven   = _____;
semaphore access = _____;
make_pizza(int size, int toppings) {
    prepare_pizza(size, toppings);
    _____
    put_into_oven();
    _____
    wait_until_done();
    _____
    take_from_oven();
    _____
} // make_pizza
```

Sketching a solution

- The two resources are already identified
  - The oven
  - Access to the oven (mutex)
- We ask the usual questions
  - And take care of avoiding ***mutex-induced deadlocks***

Solution

```
semaphore oven   = ___9__;
semaphore access = __1__;  // the mutex
make_pizza(int size, int toppings) {
    prepare_pizza(size, toppings);
    _____________________________________
    put_into_oven();
    _____________________________________
    wait_until_done();
    _____________________________________
    take_from_oven();
    _____________________________________
} // make_pizza
```

Solution (cont'd)

```
semaphore oven   = ___9__;
semaphore access = __1__;  // the mutex
make_pizza(int size, int toppings) {
    prepare_pizza(size, toppings);
    P(&oven); P(&access);_____________________
    put_into_oven();
    _____________________________________
    wait_until_done();
    _____________________________________
    take_from_oven();
    _____________________________________
} // make_pizza
```

Order matters!

Solution (cont'd)

```
semaphore oven   = ___9__;
semaphore access = __ 1__;  // the mutex
make_pizza(int size, int toppings) {
    prepare_pizza(size, toppings);
    P(&oven); P(&access); _____________________
    put_into_oven();
    V(&access);_____________________________
    wait_until_done();
    P(&access);_____________________________
    take_from_oven();
    V(&oven); V(&access);  // IN ANY ORDER!____
} // make_pizza
```