```
/*
  Uppgiften är att synkronisera koden och
  undvika alla ”busy-wait” som kan uppstå.
  Din lösning ändrar eller ersätter relevant
  kod med att använda lås och semaforer
  (conditions om du vill).

  Det finns:
  - Tio filosofer.
  - Ett runt bord med fem platser.
  - En skål mat i mitten på bordet.
  - Två servitörer som med jämna mellanrum
    fyller på skålen.
  - En tallrik per plats och en gaffel mellan
    varje plats.

  Krav:
  - Varje filosof behöver en plats, vilken som.
  - En filosof behöver just de gafflar som
    finns på vardera sida om platsen.
  - Det går inte att ta mat om skålen är tom.
  - Är skålen tom lämnar filosofen bordet
    omedelbart.
  - En filosof skall kunna äta samtidigt som de
    andra filosoferna.

  OBS: De funktionerna som ej är implementerade
       (endast deklarade) är trådsäkra.

  Tänk även på att undvika deadlock!
*/
```

```
1
2  // begin of dummy declarations that are only
3  // designed to get the code through compiler
4  typedef enum      {false, true } bool;
5  struct semaphore { int count; };
6  struct lock      { int owner; };
7  struct condition { int placeholder; };
8
9  // The five functions are thread-safe.
10 void cook_more_rations();
11 void eat_randomly_long_time();
12 void sleep_randomly_long_time();
13 void atomic_swap(void*, void*);
14 void swap(int*, int*);
15
16 void sema_init(struct semaphore*, int);
17 void sema_down(struct semaphore*);
18 void sema_up(struct semaphore*);
19
20 void lock_init(struct lock*);
21 void lock_acquire(struct lock*);
22 void lock_release(struct lock*);
23
24 void cond_init (struct condition*);
25 void cond_wait (struct condition*, struct lock*);
26 void cond_signal (struct condition*, struct lock*);
27 // end of dummy declarations
```

```
1  bool seat_taken[5];
2  bool fork_taken[5];
3  int  rations_of_food = 0;
4
5  void init() // executed before threads
6  {
7    for (int i = 0; i < 5; ++i)
8    {
9      seat_taken[i] = false;
10     fork_taken[i] = false;
11   }
12 }
13
14 void clerk_thread() // two of those
15 {
16   for ( ; ; )
17   {
18     // Add more food to table
19     rations_of_food = rations_of_food + 1;
20     cook_more_rations();
21   }
22 }
```

```
void philosopher_thread() // ten of those
{
  for ( ; ; )
  {
    int seat_no = 0;

    // Find a free seat.
    while ( seat_taken[seat_no] )
      seat_no = (seat_no + 1) % 5;
    seat_taken[seat_no] = true;

    // Grab two forks.
    int fork1 = seat_no;
    int fork2 = (seat_no + 1) % 5;

    while ( fork_taken[fork1] )
      ;
    fork_taken[fork1] = true;

    while ( fork_taken[fork2] )
      ;
    fork_taken[fork2] = true;

    // Check for food. Eat, or leave hungry.
    if (rations_of_food > 0)
    {
      rations_of_food = rations_of_food - 1;
      eat_randomly_long_time();
    }

    // Replace forks and leave seat.
    fork_taken[fork1] = false;
    fork_taken[fork2] = false;
    seat_taken[seat_no] = false;

    // Digest the food.
    sleep_randomly_long_time();
  }
}
```