Programming Exercise 1 – Melon Box Problem

In this exercise you will write a pthread program that uses pthread mutexes and condition variables to solve a simple synchronization problem.

Melon box problem: Suppose that there are multiple consumers (who come and go), one farmer and a melon box where melons are stored. The farmer places melons into the box, one at a time in regular intervals, provided that the box has less than *m* melons (*m* is a positive integer). If the box is full (i.e. it has *m* melons), then the farmer waits until the box is not full (i.e. melons have been taken from the box by consumers). When a consumer arrives, if the box contains at least one melon, then he takes one melon and leave. Otherwise, the consumer leaves immediately with nothing.

Write a program to solve this problem using **condition variables** and **mutexes** to provide synchronization between the consumers and the farmer. To assist you get started on this exercise, a program template `melon_box.c` has been written for you. You need to fill in the sections marked with dots and also write two complete thread routines `farmer_routine` and `consumer_routine`.

In `farmer_routine`, you need to add print statements to print out

- `"Farmer: the box is full containing %d melons and I'm waiting for consumers.\n"` – when the melon box is full;
- `"Farmer: I added one more melon and now the box contains %d melons.\n"` – after the farmer added a melon in the box.

In `consumer_roution`, print out

- `"Starting consumer %d.\n"` – when a consumer thread starts;
- `"Consumer %d: Oh no! the melon box is empty and I'll leave without melons!\n"` – if the box is empty;
- `"Consumer %d: I'm lucky to get one melon from %d! \n"` – the consumer get one melon when the box is not empty.

In `main` the program first asks the user to give a few parameters:

- `melon_box_capacity` – the maximum number of melons the box can hold;
- `no_of_consumers` – the total number of consumers to be created;
- `farmer_pace` – the number of seconds the farmer sleep before adding one melon to the box;
- `consumer_rate` – consumer arrival rate.

You can change these numbers and see the melons in the box and behaviours of the farmer and the consumers accordingly when running the program.

Basic functions for mutex:

```
int pthread_mutex_init(pthread_mutex_t *mutex, const
pthread_mutexattr_t *mutexattr);
```

Description:

This function initializes mutex with attributes specified by mutexattr. If mutexattr is NULL, the default mutex attributes are used. Upon successful initialization, the state of mutex becomes initialized and unlocked.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Description:

The mutex object shall be locked by calling this function. If mutex is already locked, the calling thread shall block until mutex becomes available.

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

Description:

This function is equivalent to pthread_mutex_lock(), except that if mutex is currently locked (by any thread, including the current thread), the call shall return immediately.

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Description:

This function shall release mutex. If there are threads blocked on the mutex object when pthread_mutex_unlock is called, mutex shall become available, and the scheduling policy determine which thread shall acquire mutex.

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Description:

This function destroys the mutex object.

Basic functions for condition variable:

```
int pthread_cond_init(pthread_cond_t *cond, const
pthread_condattr_t *attr);
```

Description:

This function shall initialize the condition variable cond with attributes referenced by attr. If attr is NULL, the default condition variable attributes shall be used.

```
int pthread_cond_signal(pthread_cond_t *cond);
```

Description:

This function shall unblock one of the threads that are blocked on the condition variable cond if any threads are blocked on cond.

```
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Description:

This function shall unblock all threads currently blocked on cond.

```
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t
*mutex);
```

Description:

This function shall be called with mutex locked by the calling thread. The function atomically release mutex and cause the calling thread to block on the condition variable cond;

```
   int pthread_cond_destroy(pthread_cond_t *cond);
```

Description:

This function shall destroy the condition variable `cond`;

Note: A condition variable must always be used in conjunction with a `mutex` lock. Both `pthread_cond_signal()` and `pthread_cond_wait()` should be called after `mutex` is locked.