Question 1.

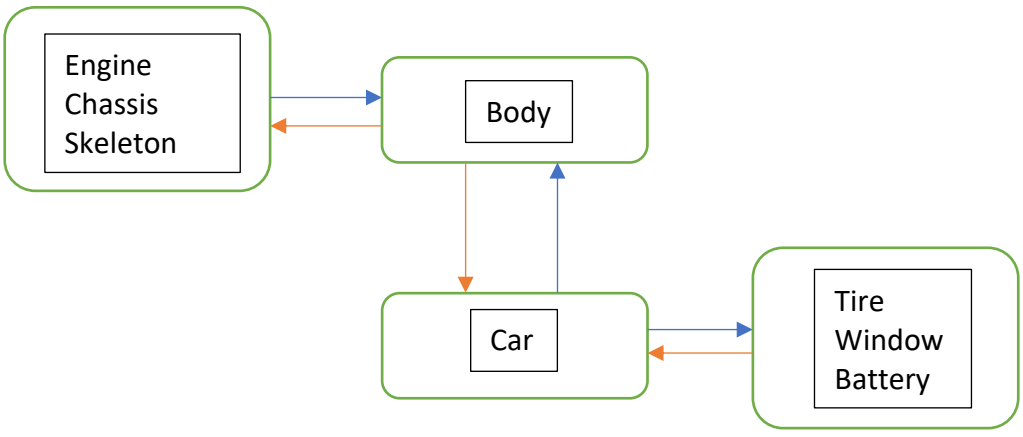
1. 16 spaces (for one car) can be considered as sufficient as there will be 16 parts that will be needed to be stored for 1 particular car if not produced in an efficient order. Car doesn't require a space.
   If car > 1
   And let m = num of cars.
   16 * m or num of robots whichever is less will be needed.
2. The increase in number of robots should increase the efficiency until a particular point and then have no effect.
3. Robots help us do multithreading therefore increase the efficiency and reduce the time whereas the number of cars increase the production requirement therefore require more time.
   Type A > Type B > Type C (Overall efficiency).

```
rm -f *.o include/*.gch
hnagra@workbench q1> ./tesla_factory.out 3 20 1 2 3
Name: Harsh Nagra        UID: 3035437707
Production goal: 3, num space: 20, num typeA: 1, num typeB: 2, num typeC: 3
====Final Report====
Num free space: 20
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 3
Production task completed! 3 cars produced.
Production time: 33.001916
hnagra@workbench q1>
```

Question 2.

Method: SEMAPHORES & QUEUES.

For every type of body part, there is a semaphore to ensure the order such that there are no deadlocks throughout the process. The number of Robots are fixed to the number of spaces.



Engine, Chassis & Skeleton cannot start working (occupying space) until the there is a Robot working on Body. This ensures that once created, these parts are picked up by the Robot working on Body (maximize storage spaces). Another variable ensures that the car is not being produced currently which could lead to deadlocks with **storage restrictions** (2 spaces).

Car cannot start working until one of the Robots is done working on the Body i.e., the body is completed. As soon as the body is completed, a robot comes in and starts working on the car. This ensures that there is no deadlock because of **shortage of robots** (2 robots).

The tire, window and battery cannot start production until a robot is making a car to ensure that these pieces are picked up as soon as they are ready to eliminate the possibility of storage getting full.

This order is achieved by using Semaphore's to keep a track of the order of production. If a condition fails at any time, that particular task in enqueued at the end of the queue rather than waiting for a spot/robot to get free and therefore, creating a deadlock.

This method ensures that there are no deadlocks but keeping an in-depth track of production requirements such that there is nothing produced in the incorrect order or amount leading to storage spaces getting filled up or all the robots getting occupied in unrelated process.

Second Method (Easy method but UNPREDICTABLE/UNRELIABLE)

1. Change the order of the queue to: Body -> Engine/Chassis/Skeleton -> Car -> Tire/Windows/Battery
2. Limiting the number of robots to number of spaces i.e., total robots <= total spaces

Also, does not cover the methods taught in course therefore not useful.

Question 3.

Key up points of my algorithm –

1. Chooses the fastest Robots (if Robots > Spaces; Priority – A, B, C).
2. Simultaneous Production (Multithreading).

The program ensures simultaneous production by using multiple robots (of course, depending on number of spaces available). Therefore, letting the parts for body execute simultaneously and the parts for car execute simultaneously.

A bottle neck is created by my algorithm while waiting for the body to be made by the car and other parts to get started which slows down the process a bit.

The number of storage spaces will have the heaviest impact on the program. As you can see below, changing the storage spaces from 17 to 2 increased the time taken by about 22 seconds.

```
hnagra@workbench q3> ./tesla_factory.out 1 17 17 0 0
Name: Harsh Nagra        UID: 3035437707
Production goal: 1, num space: 17, num typeA: 17, num typeB: 0, num typeC: 0
====Final Report====
Num free space: 17
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 1
Production task completed! 1 car produced.
Production time: 18.000776
hnagra@workbench q3> ./tesla_factory.out 1 2 17 0 0
Name: Harsh Nagra        UID: 3035437707
Production goal: 1, num space: 2, num typeA: 17, num typeB: 0, num typeC: 0
====Final Report====
Num free space: 2
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 1
Production task completed! 1 car produced.
Production time: 40.002200
```

Changing the number of robots has a similar impact on the algorithm as only two threads will be executed at the same time. 22 seconds lost.

```
hnagra@workbench q3> ./tesla_factory.out 1 17 2 0 0
Name: Harsh Nagra        UID: 3035437707
Production goal: 1, num space: 17, num typeA: 2, num typeB: 0, num typeC: 0
====Final Report====
Num free space: 17
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 1
Production task completed! 1 car produced.
Production time: 40.002310
```

When the number of cars increase, the time certainly increases as there are a greater number of tasks to be done. The effect will be linear as the second car starts after the first is completed and so on, but this process could have been improved by assigning a set of robots to every car if the number of robots increase a certain limit.

Example –
2 Cars, 4 spaces, 4 Robot A could produce the cars parallelly and take about 40 seconds to make 5 cars (Each car gets 2 robots working on it).

But as my algorithm does not account for that it takes more time than a more approach mentioned above.

```
Production time: 40.002510
hnagra@workbench q3> ./tesla_factory.out 2 4 4 0 0
Name: Harsh Nagra        UID: 3035437707
Production goal: 2, num space: 4, num typeA: 4, num typeB: 0, num typeC: 0
====Final Report====
Num free space: 4
Produced Skeleton: 0
Produced Engine: 0
Produced Chassis: 0
Produced Body: 0
Produced Window: 0
Produced Tire: 0
Produced Battery: 0
Produced Car: 2
Production task completed! 2 cars produced.
Production time: 44.002076
```

The performance is certainly sacrificed to deal with deadlocks for situations with less robots and spaces as now the car needs to wait for the body to be complete.

The production time in my algorithm can easily be predicted as we need to account for only a few things –

1. Parallel execution of Engine/Chassis/Skeleton while makeBody() is running. makeBody() only starts when last car is completed.
2. makeCar() starts right after makeBody() is complete and now Tire/Window/Battery can be produced parallelly. The production time of parallelly executed part needs to be accounted with waiting of makeBody() and makeCar().

Including the parallel execution of multiple cars as mentioned above I would also like to prioritize available robots to tasks (a greedy approach) as every robot can work to the best of its efficiency by doing the task it does the fastest and this would have a huge impact on the performance of the algorithm.

As the number of spaces and robots has an impact on my algorithm but this impact will converge as only Engine/Chassis/Skeleton and Tire/Window/Battery are the only parts executed parallel therefore not making the best use of multithreading. Whereas, an algorithm which makes multiple cars simultaneously and considers which robot is best for a particular task would perform much better. Due to my limited knowledge in C without appropriate data structures I was not able to implement this.

This assignment certainly challenged me to think about efficiency and making the most out of multithreading and other beneficial concepts taught in this course.