Question 1

```
sem_t allowed_a = 1;
sem_t allowed_b = 0;
sem_t allowed_c = 0;
```

while (1) {
  wait(allowed_a);
  cout << "a";
  signal(allowed_b);
}
	while (1) {
	  wait(allowed_b);
	  cout << "b";
	  signal(allowed_c);
	}
		while (1) {
		  wait(allowed_c);
		  cout << "c";
		  signal(allowed_a);
		}

Question 4

```
// Semaphores
sem_t ladder_available = 1;

// Mutex lock
sem_t mutex = 1;

// Count of robots
int up_robots = 0;
int down_robots = 0;

// Helper functions
void mutex_inc(int& x) {
      wait(mutex);
      x++;
      signal(mutex);
}

void mutex_dec(int& x) {
      wait(mutex);
      x--;
      signal(mutex);
}
```

```
struct robot {

      void move_up() {

            if (up_robots == 0) {
                  wait(ladder_available);
            }

            mutex_inc(up_robots);

            // .. move up code ..

            mutex_dec(up_robots);

            if (up_robots == 0) {
                  signal(ladder_available);
            }


      }

      void move_down() {

            if (down_robots == 0) {
                  wait(ladder_available);
            }

            mutex_inc(down_robots);

            // .. move down code ..

            mutex_dec(down_robots);

            if (down_robots == 0) {
                  signal(ladder_available);
            }

      }

}
```

Question 5

```
sem_t barber_sleep = 0;     // 0 -> no, 1 -> yes
int free_chairs = N;        // total free chairs available

class BarberShop

      void customer_arrives() {
            if (chairs) {
                  mutex_dec (free_chairs);
            }
            return;
      }

      void wakeup_barber() {
            wait(barber_sleep);
      }

      void sleep_barber() {
            signal(barber_sleep);
      }

      void serve_customer() {
            mutex_inc(free_chairs);
      }

      BarberShop() {

            while (1) {
                  sleep_barber();
                  // Here we can add code for customer arrival
                  if (free_chairs < N) {
                        wakeup_barber();
                        while (free_chairs != N)
                              serve_customer();
                  }
                  else {
                        wakeup_barber();
                  }
            }
      }
```

Question 2

This code is wrong due to the improper signaling and waiting of the barrier semaphore as well as the calling of function before the checking and count. It results in a deadlock.

An appropriate solution would be:

```
int n = ...
int count = 0;
sem_t mutex = 1;
sem_t barrier = 0;

wait(mutex);
     ++count;
signal(mutex);

if (count == n)
     signal(barrier);

wait(barrier);
signal(barrier);

foo();
doSomething();
```

Question 3

```
void push(int x) {

     if (top == max)
          wait(stack_has_space);

     wait(mutex);
          a[top] = x;
          ++top;
     signal(mutex);
     wait(stack_has_space);
     signal(stack_is_filled);
}
```

```
int pop() {

      if (top == 0)
            wait(stack_is_filled);

      wait(mutex);
            --top;
            return a[top + 1];
      signal(mutex);
      wait(stack_is_filled);
      signal(stack_has_space);

}
```