COSC 3360/6310 — FUNDAMENTALS OF OPERATING SYSTEMS
ASSIGNMENT #3: THE DRAWBRIDGE
Due Monday, May 1, 2023 at 11:59:59.99 pm

OBJECTIVE
You are to learn how to use POSIX threads and POSIX advanced synchronization feature

THE PROBLEM
A narrow drawbridge is typically in its down position and requires cars to cross it one at a time. From time to time, it needs to be raised to let a ship go through. As the event is relatively infrequent, the bridge is always brought back to the down position once the shipped has gone through the channel.

YOUR PROGRAM
All your program inputs will be read from the—redirected—standard input.

Your first line on input will contain one string followed by two integers being:

1. The keyword "`Bridge`",
2. The time it takes to raise the bridge,
3. The time to lower it down.

All remaining lines will describe either a car or a ship arriving at the bridge and will contain two strings followed by two integers being:

1. The keyword "`Car`" or the keyword "`Ship`"
2. A car license plate number or the name of a ship,
3. The time elapsed since the arrival of the previous car or ship,
4. The time it will take for the car to cross the bridge or the ship to cross the channel.

One possible set of input could be:

```
Bridge 1 1
Car STOL3N 0 2
Car HIOFCR 1 2
Ship Alicia 1 2
Car 2ZBEACH 2 2
```

For obvious reasons, these minutes will be represented by seconds in our simulation.

Your program should print out a descriptive message including the car license plate each time a car:

1. Arrives at the bridge,
2. Goes on the bridge,
3. Leaves the bridge.

as in:

```
Car HIOFCR arrives at the bridge.
Car HIOFCR goes on the bridge.
Car HIOFCR leaves the bridge.
```

It should print similar descriptive message including the ship's name each time a ship:

1. Arrives at the bridge,
2. Goes through the channel,
3. Leaves the area.

as in:

```
Ship Alicia arrives at the bridge.
Bridge is closed to car traffic.
Bridge can now be raised.
Ship Alicia goes through the channel.
Ship Alicia is leaving.
Bridge can now accommodate car traffic.
```

At the end your program should display the total number of cars and ships that went through as in:

```
3 car(s) crossed the bridge.
1 ship(s) went through the channel.
```

NON-DETERMINISTIC OUTPUTS
You will notice your program will produce non-deterministic outputs each time two events happen at the same time, say, when a visitor arrives just when another leaves the platform. These non-deterministic outputs occur because we have no control on the way our pthreads are scheduled. The sole way to guarantee a deterministic output is to come with input data that generate schedules where each event happens at a different unique time.

PTHREADS
1. Don't forget the pthread include:

   ```
   #include <pthread.h>
   ```

   and the **`-lpthread`** library option in your compilation options
2. All variables that will be shared by all threads must be declared outside of any function as in:

   ```
   static int nCars, nShips;
   ```

3. If you want to pass any data to your thread function, you should declare it **`void`** as in:

   ```
   void *car(void *arg) {
       vData = (struct cData) arg;
       …
   } // car
   ```

You ***must immediately copy*** the contents of **`cData`** into a local variable**.**

Since some C++ compilers treat the cast of a **`void`** into anything else as a fatal error, you might want use the flag **`-fpermissive`**.

4. To start a thread that will execute the visitor function and pass to it an integer value use:

```
pthread_t tid;
…
pthread_create(&tid, NULL,
       car, (void *) cData);
```

Since you have to pass a string and two integers to the **`car`** function or to the **`ship`** function, you should have put them into a structure.

5. To terminate a given thread from inside the thread function, use:

```
pthread_exit((void*) 0);
```

Otherwise, the thread will terminate with the function.

6. If you have to terminate another thread function, you many use:

```
#include <signal.h>
pthread_kill(pthread_t tid,
             int sig);
```

Note that **`pthread_kill()`** is a dangerous system call because its default action is to immediately terminate the target thread even when it is in a critical section. The safest alternative to kill a thread that repeatedly executes a loop is through a shared variable that is periodically tested by the target thread. ***You will not have to use it in your program.***

7. To wait for the completion of a specific thread use:

```
pthread_join(tid, NULL);
```

Note that the pthread library has no way to let you wait for an unspecified thread and do the equivalent of:

```
for (i = 0; i < nChildren; i++)
    wait(0);
```

Your main thread will have to keep track of the thread id's of all the threads of all the threads it has created:

```
pthread_t tid[MAXTHREADS];
. . .
for (i=0; i< nThreads; i++)
    pthread_join(tid[i], NULL);
```

PTHREAD MUTEXES

1. To be accessible from all threads pthread mutexes must be declared outside of any function:

```
static pthread_mutex_t bridge;
```

2. To create a mutex use:

```
pthread_mutex_init(&access, NULL);
```

Your mutex will be automatically initialized to one.

3. To acquire the lock for a given resource, do:

```
pthread_mutex_lock(&bridge);
```

4. To release your lock on the resource, do:

```
pthread_mutex_unlock(&bridge);
```

PTHREAD CONDITION VARIABLES

1. The easiest way to create a condition variable is:

```
static pthread_cond_t carsCanGo=
    PTHREAD_COND_INITIALIZER;
```

2. Your condition waits must be preceded by a successful lock request on the mutex that will be passed to the wait:

```
pthread_mutex_lock(&bridge);
while (bridgeStatus != CARSCANGO)
    pthread_cond_wait(&carsCanGo,
     &bridge);
…
pthread_mutex_unlock(&bridge);
```

3. To avoid unpredictable scheduling behavior, the thread calling **`pthread_cond_signal()`** must ***own*** the mutex that the thread calling **`pthread_cond_wait()`** had specified in its call:

```
pthread_mutex_lock(&bridge);
…
pthread_cond_signal(&carsCanGo);
pthread_mutex_unlock(&bridge);
```