```cpp
/*
CPSC 457
Threads, mutex, condition variable, lock_guard, unique_lock, and semaphore examples
*/
//================================================================================
/*
Exercise: The following code uses two threads to find a digit in a range of numbers.
Compile and check the run time.

How can this be made more efficient?

Source: Sina Keshvadi (2019)
*/
#include <iostream>
#include <thread>
#include <math.h>

using namespace std;

int flag = 1;
long range = 10000000000;

void search(long first, long end, long x) {
    for (long i=first; i<=end; i++){
        if (i==x){
            cout<<"Found it! i = " << i <<endl;
        }
    }
}

int main(int argc, char const *argv[]) {
    long x = 10;

    thread t1(search, 1, ceil(range/2), x);
    thread t2(search, ceil(range/2), range, x);

    t1.join();
    t2.join();
    return 0;
}


//================================================================================
/*
Example 1: using condition variable with mutex and unique_lock (lock guard). Use notify_all function.
Source: http://www.cplusplus.com/reference/condition_variable/condition_variable/
*/
// condition_variable example
#include <iostream>           // std::cout
#include <thread>             // std::thread
#include <mutex>              // std::mutex, std::unique_lock
#include <condition_variable> // std::condition_variable
```

```
53
54 using namespace std;
55
56 mutex mtx;
57 condition_variable cv;
58 bool ready = false;
59
60 void print_id (int id) {
61   unique_lock<mutex> lck(mtx);    // unique_lock
62   while (!ready) cv.wait(lck);            // wait
63   // ...
64   cout << "thread " << id << '\n';
65 }
66
67 void go() {
68   unique_lock<mutex> lck(mtx);
69   ready = true;
70   cv.notify_all();                       // signal: unblock all threads
   currently waiting for this condition
71 }
72
73 int main ()
74 {
75   thread threads[10];
76   // spawn 10 threads:
77   for (int i=0; i<10; ++i)
78     threads[i] = thread(print_id,i);
79
80   cout << "10 threads ready to race...\n";
81   go();                       // go!
82
83   for (auto& th : threads) th.join();
84
85   return 0;
86 }
87
88
89 //
   ================================================================================
   ==============
90 /*
91 Example 2: using condition variable w/ Mutex and unique_lock. Use notify_one
   function.
92 Source:
   http://www.cplusplus.com/reference/condition_variable/condition_variable/noti
   fy_one/
93 */
94 // condition_variable::notify_one
95 #include <iostream>           // std::cout
96 #include <thread>             // std::thread
97 #include <mutex>              // std::mutex, std::unique_lock
98 #include <condition_variable> // std::condition_variable
99
100 using namespace std;
101
102 mutex mtx;
103 condition_variable produce,consume;
104
105 int cargo = 0;     // shared value by producers and consumers
106
```

```
107 void consumer () {
108   /*
109   unique_lock: an object that manages a mutex object
110   - unique_lock calls unlock on the mutex in its destructor.
111   Benefit: in the case you leave the scope that unique_lock is defined in
    (maybe due to an exception being thrown), you can be sure that the mutex will
    unlock.
112   */
113   unique_lock<mutex> lck(mtx);              // create unique_lock with mtx
    and lock mtx
114   // if cargo is 0, then wait until there's cargo to consume
115   while (cargo==0) consume.wait(lck);       // cv consume calls wait on lck
    which has the mutex mtx.
116   cout << cargo << '\n';
117   cargo=0;
118   produce.notify_one();    // signal (unblock) all threads waiting on the
    condition variable produce.
119 }
120
121 void producer (int id) {
122   unique_lock<mutex> lck(mtx);   // create unique_lock with mtx and lock mtx

123   // if cargo isn't empty, wait for it to be consumed
124   while (cargo!=0) produce.wait(lck);
125   cargo = id;
126   consume.notify_one();         // signal (unblock) all threads waiting on
    the consume condition variable.
127 }
128
129 int main ()
130 {
131   // create 10 consumer threads, and 10 producer threads
132   thread consumers[10],producers[10];
133   for (int i=0; i<10; ++i) {
134     consumers[i] = thread(consumer);        // consumer thread calls
    consumer() function
135     producers[i] = thread(producer,i+1);    // producer thread calls
    producer() function
136   }
137
138   // join them back:
139   for (int i=0; i<10; ++i) {
140     producers[i].join();
141     consumers[i].join();
142   }
143
144   return 0;
145 }
146
147
148 //
    ================================================================================
    ==============
149 /*
150 Example 3 using a semaphore to synchronize 15 threads.
151
152 Source: https://austingwalters.com/multithreading-semaphores/
153 */
154 #include <iostream>
155 #include <thread>
```

```cpp
#include <mutex>
#include <condition_variable>

std::mutex mtx;             // mutex for critical section
std::condition_variable cv; // condition variable for critical section
bool ready = false;         // Tell threads to run
int current = 0;            // current count

/* Prints the thread id / max number of threads */
void print_num(int num, int max) {

  std::unique_lock<std::mutex> lck(mtx);
  while(num != current || !ready){ cv.wait(lck); }      // *** Key line that prevented a race condition ***
  current++;
  std::cout << "Thread: ";
  std::cout << num + 1 << " / " << max;
  std::cout << " current count is: ";
  std::cout << current << std::endl;

  /* Notify next threads to check if it is their turn */
  cv.notify_all();
}

/* Changes ready to true, and begins the threads printing */
void run(){
  std::unique_lock<std::mutex> lck(mtx);
  ready = true;
  cv.notify_all();
}

int main (){

  int threadnum = 15;
  std::thread threads[15];

  /* spawn threadnum threads */
  for (int id = 0; id < threadnum; id++)
    threads[id] = std::thread(print_num, id, threadnum);

  std::cout << "\nRunning " << threadnum;
  std::cout << " in parallel: \n" << std::endl;

  run(); // Allows threads to run

  /* Merge all threads to the main thread */
  for(int id = 0; id < threadnum; id++)
    threads[id].join();

  std::cout << "\nCompleted semaphore example!\n";
  std::cout << std::endl;

  return 0;
}
```