Tema: Una Barriera Ciclica è un oggetto di sincronizzazione che permette a N thread (con N specificato all'atto della costruzione dell'oggetto) di attendersi. Tale oggetto dispone di un solo metodo "void attendi()" la cui invocazione blocca il thread chiamante fino a che altri N-1 thread non risultano bloccati insieme ad esso. Quando il numero di thread bloccati raggiunge N, tutti thread si sbloccano e il metodo attendi() ritorna. Ulteriori chiamate al metodo attendi() si comportano analogamente: le prime N-1 invocazioni si bloccano, all'arrivo della successiva invocazione tutti i thread si sbloccano e i metodi ritornano. Si implementi una classe C++ che implementi tale comportamento, usando le primitive Win32 (per chi ha frequentato prima del 2013) o la libreria standard C++ (per chi ha frequentato nel 2013).

```cpp
#include <iostream>
#include <thread>
#include <chrono>

#include <condition_variable>
#include <functional>


class Barriera_Ciclica {

    std::mutex m;
    std::condition_variable cv;
    int N, cont;
    bool blocked, allwakedup;

public:
    Barriera_Ciclica(int n)
            : N(n), cont(0), blocked(true), allwakedup(true){}

    void attendi(int n){
        std::unique_lock ul(m);
        std::cout << "aspetto all'entrata, t" << n << std::endl;
        cv.wait(ul, [this]() { return allwakedup; });
        std::cout << "passato t" << n << std::endl;
        cont ++;
        if(cont!=N) {
            cv.wait(ul, [this]() { return !blocked; });
        }
        else {
            blocked=false;
            allwakedup=false;
            std::cout << "risveglio tutti, sono t" <<n<<std::endl;
            cv.notify_all();
        }
        //il 5 mantiene il lock
        cont--;
        if(cont==0){
            std::cout << "ultimo thread a svegliarsi, t "<< n << std::endl;
            blocked=true;
            allwakedup = true;
            cv.notify_all();
        }
    }//lock rilasciato, gli altri si possono svegliare
};

int main(){
    Barriera_Ciclica bc (5);
    using namespace std::chrono_literals;

    std::thread t1([&](){
        for(int i=0; i<3; i++) {
            bc.attendi(1);
        }

    });

    std::thread t2([&](){
        for(int i=0; i<3; i++) {
            std::this_thread::sleep_for(2000ms);
            bc.attendi(2);
```

```
61          }
62      });
63
64      std::thread t3([&](){
65          for(int i=0; i<3; i++) {
66              bc.attendi(3);
67          }
68      });
69
70      std::thread t4([&](){
71          for(int i=0; i<3; i++) {
72              std::this_thread::sleep_for(2000ms);
73              bc.attendi(4);
74          }
75      });
76      std::thread t5([&](){
77          for(int i=0; i<3; i++) {
78              bc.attendi(5);
79          }
80      });
81
82      if ( t1.joinable()) t1.join();
83      if ( t2.joinable()) t2.join();
84      if ( t4.joinable()) t4.join();
85      if ( t3.joinable()) t3.join();
86      if ( t5.joinable()) t5.join();
87
88      return 0;
89  }
```