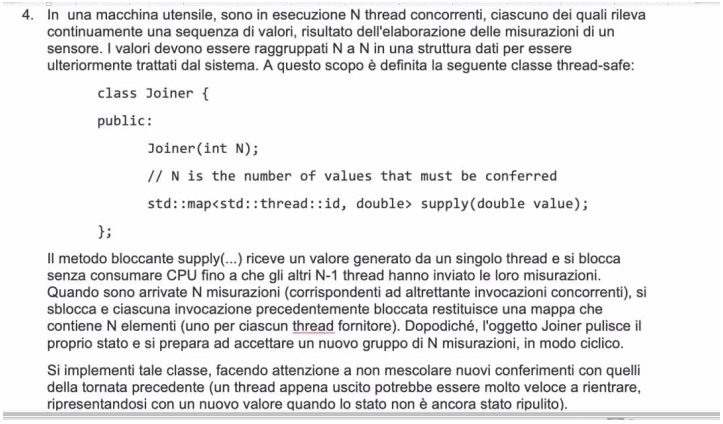

4. In una macchina utensile, sono in esecuzione N thread concorrenti, ciascuno dei quali rileva continuamente una sequenza di valori, risultato dell'elaborazione delle misurazioni di un sensore. I valori devono essere raggruppati N a N in una struttura dati per essere ulteriormente trattati dal sistema. A questo scopo è definita la seguente classe thread-safe:

```
class Joiner {
public:
    Joiner(int N);
    // N is the number of values that must be conferred
    std::map<std::thread::id, double> supply(double value);
};
```

Il metodo bloccante supply(...) riceve un valore generato da un singolo thread e si blocca senza consumare CPU fino a che gli altri N-1 thread hanno inviato le loro misurazioni. Quando sono arrivate N misurazioni (corrispondenti ad altrettante invocazioni concorrenti), si sblocca e ciascuna invocazione precedentemente bloccata restituisce una mappa che contiene N elementi (uno per ciascun thread fornitore). Dopodiché, l'oggetto Joiner pulisce il proprio stato e si prepara ad accettare un nuovo gruppo di N misurazioni, in modo ciclico.

Si implementi tale classe, facendo attenzione a non mescolare nuovi conferimenti con quelli della tornata precedente (un thread appena uscito potrebbe essere molto veloce a rientrare, ripresentandosi con un nuovo valore quando lo stato non è ancora stato ripulito).

```cpp
#include <iostream>
#include <map>
#include <thread>
#include <condition_variable>
#include <mutex>

class Joiner {
    std::condition_variable cv;
    std::mutex m;
    int count;
    int N;
    std::map<std::thread::id,double>  values;
    bool entering;
public:
    Joiner(int N): N(N), count(0), entering(true) {}// N is the number of values that must be conferred

    std::map<std::thread::id, double> supply(double value) {
        std::unique_lock<std::mutex> ul(m);
        cv.wait(ul, [this](){return entering;});
        values[std::this_thread::get_id()] = value; //qui cambio il ritorno
        count++;
        if (count<N) {
            cv.wait(ul, [this](){return !entering;});
        } else {
            cv.notify_all();
            entering=false;
        }
        count--;
        if (count==0) { //ultimo pulisce e torna anche come gli altri
            entering = true;
            cv.notify_all();
            std::map<std::thread::id,double> values1(values);
            values.clear();
            return values1;
        }
        return values; //tutti ritornano stessa cosa
    }
};

int main() {
    const int N = 4;
    Joiner joiner(N);

    int iter = 10000;
    auto f = [&joiner,N,iter](bool print) {
        for (int i=0; i<iter; i++) {
            auto m = joiner.supply(i);
            //assert(m.size() == N);
            for (auto p: m) {
                if (p.second!=i) std::cout<<"Expected "<<i<<" found "<<p.second<<std::endl; //in caso di fail
                //assert(p.second==i);
            }
            if(print) {
                std::cout<<i<<" ";
                if (i%10==9) std::cout<<std::endl;
            }
        }
    };
```

```
59
60      std::thread threads[N];
61      for (int i=0; i<N; i++)
62          threads[i]=std::thread(f, i==0); //solo il primo stampa
63
64          for (auto & thread : threads)//fine
65          thread.join();
66      return 0;
67  }
68
```