```
/* FILE: hanoi.cpp   last change: 27-Aug-2012   author: Romeo Rizzi
 * a solver for problem 2 (hanoi) in the 03-09-2012 exam in Algorithms
 */

#define NDEBUG // NDEBUG disabilita tutte le assert se definita prima di include
re <cassert>
#include <cassert>

#ifndef NDEBUG
#  include <iostream>  // non voglio includerla quando Non compilo in modalita'
DEBUG
#endif
#include <fstream>

using namespace std;

const int MAX_N = 1000;
int n;   // number of disks: the disks are numbered from 1 (the smallest) to n (
the biggest)
int cpd[MAX_N +1], tpd[MAX_N +1]; //current and target position of each disk

ifstream fin;
ofstream fout;

bool isTower(int n, int pole) {
  while( n > 0 )
    if( cpd[n--] != pole ) return false;
  return true;
}

void spostaDisco(int n, int from, int to) {
  assert( n > 0 );
  assert( cpd[n] == from );
  assert( isTower(n-1, 6-from-to) );
  fout << "Sposta il disco " << n << " dal piolo " << from << " al piolo " << to << endl;
  cpd[n] = to;
}

void spostaTorre(int n, int from, int to, int aux) {
  assert( n >= 0 ); if( n <= 0 ) return;
  assert( isTower(n, cpd[n]) );
  spostaTorre(n-1, from, aux, to);
  spostaDisco(n, from, to);
  spostaTorre(n-1, aux, to, from);
}

void creaTorre(int n, int pole) {
  assert( n >= 0 ); if( n <= 0 ) return;
  if( cpd[n] == pole )  creaTorre(n-1, pole);
  else {
    int aux = 6-cpd[n]-pole;
    creaTorre(n-1, aux);
    spostaDisco(n, cpd[n], pole);
    spostaTorre(n-1, aux, pole, 6 -aux -pole);
  }
}

void go_from_current_to_target_configuration() { // example of tail recursion
  int boop = n;  // name of the biggest out of place disk
  while( (boop > 0) & ( cpd[boop] == tpd[boop] ) )  boop --; // total time O(n^2
), negligiable
  if( boop == 0 )  return; // we exit only here, when all disks are in place
  int aux = 6 -cpd[boop] -tpd[boop];
  creaTorre( boop-1, aux);
  spostaDisco(boop, cpd[boop], tpd[boop]);   // at least one step has been taken
  (finiteness)
  go_from_current_to_target_configuration(); // assures we reach the target conf
. (correctness)
} // also optimality can be proven, see hint3.txt
```



```
int main() {
  fin.open("input.txt"); assert(fin);
  fout.open("output.txt"); assert(fout);
  fin >> n; int disk;
  for(int p = 1; p <= 3; p++)
    for(fin >> disk; disk; fin >> disk) cpd[disk] = p;
  for(int p = 1; p <= 3; p++)
    for(fin >> disk; disk; fin >> disk) tpd[disk] = p;
  /* if the input file was in compact form:
     for(int i = 1; i <= n; i++)  fin >> cpd[i];
     for(int i = 1; i <= n; i++)  fin >> tpd[i];   */
  go_from_current_to_target_configuration();
  fin.close(); fout.close();
  return 0;
}
```