```cpp
#include <cmath>
#include <iostream>
#include <iterator>
#include <algorithm>
#include <vector>

template <typename T = long>
class PrimeGeneratorIterator {
  public:
    using iterator_category = std::input_iterator_tag;
    using value_type = T;
    using difference_type = T;
    using pointer = T*;
    using reference = T&;

    PrimeGeneratorIterator() : currentPrime(2) {}
    PrimeGeneratorIterator(T initialValue) : currentPrime(initialValue){
      if (!isPrime(initialValue)){
        generateNextPrime();
      }
    }

    bool operator<(const PrimeGeneratorIterator& other) {
      return this->currentPrime < other.currentPrime;
    }

    bool operator>(const PrimeGeneratorIterator& other) {
      return this->currentPrime > other.currentPrime;
    }

    bool operator==(const PrimeGeneratorIterator& other) {
      return this->currentPrime == other.currentPrime;
    }

    bool operator!=(const PrimeGeneratorIterator& other) {
      return this->currentPrime != other.currentPrime;
    }

    reference operator*() {
      return currentPrime;
    }

    pointer operator->() {
      return &currentPrime;
    }

    //Pre-increment operator
    PrimeGeneratorIterator& operator++() {
      generateNextPrime();
      return *this;
    }

    //Post-increment operator
    PrimeGeneratorIterator operator++(int) {
      PrimeGeneratorIterator pgi(*this);
      ++*this;
      return pgi;
    }

  private:
    T currentPrime;

    void generateNextPrime() {
      while(!isPrime(++currentPrime));
    }

    bool isPrime(T num){
      if (num <= 1) {
        return false;
```

```
      }

      if (num == 2) {
        return true;
      }

      if (num % 2 == 0){
        return false;
      }

      for (int i = 3; i <= std::sqrt(num); i += 2){
        if (num % i == 0){
          return false;
        }
      }

      return true;
    }

};

using namespace std;

int main() {
  PrimeGeneratorIterator<> p_gen;
  PrimeGeneratorIterator<> end(1000);

  copy(p_gen, end, std::ostream_iterator<long>(std::cout, " "));

  cout << endl;

  return 0;
}
```