```cpp
#ifndef _ARRAY_HPP
#define _ARRAY_HPP

#include <iostream>
#include <initializer_list>


template<typename T, const unsigned N>
class Array {

 T data[N];


 void copy(const Array<T, N> &other) {
  for ( unsigned i=0; i<N; i++ ) { this->data[i] = other[i]; }
 }


 // Class Iterator

 class Iterator {

  T* ptr;

 public:

  Iterator(T* ptr): ptr{ptr} {}

  T &operator*() { return *(this->ptr); }
  Iterator &operator++() { this->ptr++; return *this; }
  Iterator &operator--() { this->ptr--; return *this; }

  Iterator operator+(int val) const { return Iterator(this->ptr + val); }
  Iterator operator-(int val) const { return Iterator(this->ptr - val); }

  auto operator<=>(const Iterator& other) const { return this->ptr <=> other.ptr; }
  bool operator!=(const Iterator& other) const { return this->ptr != other.ptr; }

 };

 class ConstIterator {

  T* ptr;

 public:

  ConstIterator(T* ptr): ptr{ptr} {}

  const T &operator*() const { return *(this->ptr); }
  ConstIterator &operator++() { this->ptr++; return *this; }
  ConstIterator &operator--() { this->ptr--; return *this; }

  ConstIterator operator+(int val) const { return ConstIterator(this->ptr + val); }
  ConstIterator operator-(int val) const { return ConstIterator(this->ptr - val); }

  auto operator<=>(const ConstIterator& other) const { return this->ptr <=> other.ptr; }
  bool operator!=(const ConstIterator& other) const { return this->ptr != other.ptr; }

 };


public:

 // Member Function

 constexpr Array()=default;
 constexpr explicit Array(T val) {
  for ( unsigned i=0; i<N; i++ ) { data[i] = val; }
 }
 constexpr explicit Array(std::initializer_list<T> args) {
  if (args.size() > N) { perror("Too many arguments"); exit(1); }
  if (args.size() < N) { perror("Arguments are missing"); exit(1); }
  int i=0; for ( auto arg : args ) { data[i] = arg; i++; }
 }
 constexpr Array(const Array<T, N> &other) { this->copy(other); }
 constexpr void operator=(const Array<T, N> &other) { this->copy(other); }


 // Element access

 T& at(int i) {
  if ( i < 0 or i >= N ) { perror("Bad index in Array"); exit(1); }
  return data[i];
 }
 T& operator[](int i) { return data[i]; }

 const T& at(int i) const {
  if ( i < 0 or i >= N ) { perror("Bad index in Array"); exit(1); }
  return data[i];
 }
 const T& operator[](int i) const { return data[i]; }


 // Iterators

 Iterator begin() { return Iterator(this->data); }
 Iterator end() { return Iterator(this->data + N); }

 ConstIterator cbegin() { return ConstIterator(this->data); }
 ConstIterator cend() { return ConstIterator(this->data + N); }

 Iterator rbegin() { return Iterator(this->data + N-1); }
 Iterator rend() { return Iterator(this->data - 1); }

 ConstIterator crbegin() { return ConstIterator(this->data + N-1); }
 ConstIterator crend() { return ConstIterator(this->data - 1); }


 // Capacity

 int size() const { return N; }


 // Debug

 void print() const {
  std::cout << "(";
  if (N == 0) { std::cout << "]"; return; }
  for (unsigned i=0; i<N-1; i++) {
   std::cout << this->data[i] << ", ";
  } std::cout << this->data[N-1] << ")" << std::endl;
 }

};

#endif
```