```
// Sam Borick <sb205@uakron.edu>
//
// list.hpp: Definition of the list template and its interface.

#ifndef LIST_HPP
#define LIST_HPP

// DO NOT REMOVE THIS.
// DO NOT INCLUDE <cassert>. This file contains a replacement for
// assert() that works with the testing module.
#include "test.hpp"

#include <initializer_list>

template<typename T>
struct Node{
  T value;
  Node* next;
  Node* prev;

  Node() = default;
  Node(T t, Node* n, Node* p)
  :value(t), next(n), prev(p){}

  Node & operator=(const Node & N){
    value = N.value;
    next = N.next;
    prev = N.prev;
    return *this;
  }
};


// A doubly linked list.
template<typename T>
struct List
{
  List(std::initializer_list<T>);
  Node<T>* head;
  Node<T>* tail;
  size_t len;

  List()
  :head(nullptr), tail(nullptr), len(0){}

  //copy constructor
  List(const List<T> & l){
    Node<T>* p = l.head;
    head = p;
    Node<T>* past;
    while(p){
      Node<T>* q = new Node<T>;
      *q = *p;
      if(head == p){
        head = q;
      }else{
        past->next = q;
        q->prev = past;
      }
      past = q;
      p = q->next;
    }
    tail = past;
  }


  //copy assignment
  List & operator=(const List & L){
```



```
    List temp = L;
    swap (*this, temp);
    return *this;
  }

  Node<T>* getHead()const{
    return head;
  }

  Node<T>* getTail()const{
    return tail;
  }

  void swap(List a, List b){
    Node<T>* temp1 = a.getHead();
    Node<T>* temp2 = a.getTail();

    a.head = b.getHead();
    a.tail = b.tail;
    b.head = temp1;
    b.tail = temp2;
  }

  void push_back(T t){
    Node<T>* p = new Node<T>(t, nullptr, tail);
    if(tail != nullptr){
      tail->next = p;
    }else{
      head = p;
    }
    tail = p;
    ++len;
  }

  void pop_back(){
    assert(tail != nullptr);
    if(head == tail){
      delete head;
      head = nullptr;
      tail = nullptr;
    }else{
      Node<T>* temp = tail;
      tail = tail->prev;
      tail->next = nullptr;
      delete temp;
    }
    --len;
  }

  void push_front(T t){
    Node<T>* p = new Node<T>(t, head, nullptr);
    if(head!= nullptr){
      head->prev = p;
    }else{
      tail = p;
    }
    head = p;
    ++len;
  }

  void pop_front(){
    assert(head != nullptr);
    if(head == tail){
      delete head;
      head = nullptr;
      tail = nullptr;
    }else{
      Node<T>* temp = head;
      head = head->next;
```

```
      head->prev = nullptr;
      delete temp;
    }
    --len;
  }

  int compare(const List<T> & b)const{
    Node<T>* temp1 = head;
    Node<T>* temp2 = b.head;
    while (temp1 != tail && temp2 != b.tail) {
      if (temp1->value < temp2->value)
        return -1;
      if (temp2->value < temp1->value)
        return 1;
    temp1 = temp1->next;
    temp2 = temp2->next;
    }
    if (temp1 == tail) {
      if (temp2 != tail)
        return -1; // [first1, last1) is a prefix of [first2, last2)
      else
        return 0;  // [first1, last1) and [first2, last2) are equivalent
    }
    else {
      return 1;    // [first2, last1) is a prefix  of [first1, last1)
    }
  }

  size_t size()const{
    return len;
  }

  bool empty()const{
     return (head == nullptr && tail == nullptr);
  }

  T front(){
    return head->value;
  }

  T back(){
    return tail->value;
  }

  void clear(){
    Node<T> *p = head;
    while(p){
      Node<T>* q = p-> next;
      delete p;
      p = q;
    }
    head = tail = nullptr;
  }

  ~List<T>(){
    clear();
  }
};


// TODO: Be sure to perform necessary initialization of members.
template<typename T>
List<T>::List(std::initializer_list<T> list)
{
  head = nullptr;
  tail = nullptr;
  len = 0;
  for (T const& elem : list)
    push_back(elem);
```



```
}

template<typename T>
bool operator==(List<T> const& a, List<T> const& b){
  Node<T>* temp1 = a.head;
  Node<T>* temp2 = b.head;
  while (temp1 != a.tail && temp2 != b.tail){
    if(temp1->value != temp2->value){
      return false;
    }
    temp1 = temp1->next;
    temp2 = temp2->next;
  }
  return true;
}

template<typename T>
bool operator !=(List<T> const& a, List<T> const & b){
  return !(a==b);
}

template<typename T>
bool operator < (const List<T> & a, const List<T> & b){
  return (a.compare(b) == -1);
}

template<typename T>
bool operator > (const List<T> & a, const List<T> & b){
  return (a.compare(b) == 1);
}

template<typename T>
bool operator <= (const List<T> & a, const List<T> & b){
  return !(a>b);
}

template<typename T>
bool operator >= (const List<T> & a, const List<T> & b){
  return !(a<b);
}

#endif
```

```
// Sam Borick <sb205@uakron.edu>
//

#include "list.hpp"
```