```cpp
/*
    Li, Gellert

    CS A250
    28th April, 2018

    Lab 11
*/

#include <iostream>
#include <string>
#include <vector>
#include <list>

using namespace std;

// Declaration function printVector.
// The function passes a vector and prints all
// the elements on one line, separated by a space.
// Use an iterator and a FOR loop.
void printVector(const vector<int> &v);

// Declaration function printList.
// The function passes a list and prints all
// the elements on one line, separated by a space.
// Use an iterator and a WHILE loop.
void printList(const list<int> &aList);


int main()
{

    /***************************************************************************
            VECTORS
    ***************************************************************************/
    cout << "  ***  STL VECTOR CLASS  ***  \n\n";

    // Use the default constructor to declare an integer vector v1.
    vector<int> v1;

    // void push_back (const value_type& val);
    // Use function push_back to insert the following values in v1: 12, 73, 41,
    // 38, 25, 56, an 63 in this order.

    v1.push_back(12);
    v1.push_back(73);
    v1.push_back(41);
    v1.push_back(38);
    v1.push_back(25);
    v1.push_back(56);
    v1.push_back(63);

    // size_type size() const noexcept;
    // Create a variable of type int named sizeV1 and store the size of the vector.
    // Use function size to retrieve the size of the vector.
    // Make sure you cast the return value of the function size to the appropriate type.
    int sizeV1 = v1.size();

    // Use a FOR loop to print out the vector.
    // Do NOT use an iterator.
    for (auto i : v1) {
        cout << i << " ";
    }
    cout << endl;

    //void clear() noexcept;
    // Call the function clear on vector v1.
    v1.clear();

    // size_type size() const noexcept;
    // Call function size to print the size of v1.
    cout << v1.size() << endl;

    // size_type capacity() const noexcept;
    // Call function capacity to output the capacity of v1.
    cout << v1.capacity() << endl;

    // Create an array of integers containing: 10,11,12,13,14,15,16,17,18,19
    int arr[] = { 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 };

    // Use the default constructor to declare an integer vector v2.
    vector<int> v2;

    // void assign (InputIterator first, InputIterator last);
    // Use function assign to copy elements 12, 13, 14, 15, and 16 in v2.
    // One statement only.
    v2.assign(arr + 2, arr + 7);

    // Call the function printVector to print v2.
    printVector(v2);

    // const_reference back() const;
    // Use the function back output the last element in the vector
    // (Notice that the back function returns a reference.)
    cout << v2.back() << endl;

    // Use the default constructor to declare an integer vector v3.
    vector<int> v3;

    // void assign (size_type n, const value_type& val);
    // Use function assign to insert the values 7, 7, 7, 7, and 7.
    // One statement only.
    v3.assign(5, 7);

    // Call the function printVector  to print v3.
    printVector(v3);

    // const_reference at(size_type n) const;
    // Use function at to replace the middle element with 100.
    // (Notice that the at function returns a reference.)
    v3.at(v3.size() / 2) = 100;

    // Call the function printVector to print v3.
    printVector(v3);

    // vector (const vector& x);
    // Use the copy constructor to create a new vector v4 with the
    // same elements of v3.
    vector<int> v4 = v3;

    // Call the function printVector to print v4.
    printVector(v4);

    // Create an iterator iterVector4 to point to the first element of v4.
    vector<int>::iterator iterVector4 = v4.begin();

    // Create an iterator iterVector2 to point to the second element of v2.
    vector<int>::iterator iterVector2 = v2.begin() + 1;

    // iterator insert (const_iterator position, InputIterator first,
      InputIterator last);
    // Use function insert to insert the second, third, and fourth element
    // of v2 as the first, second, and third element of v4.
    // (Notice that the insert function returns an iterator,
    //   but if we do not intend to use it, we can ignore it.)
    v4.insert(iterVector4, iterVector2, iterVector2 + 3);

    // Call the function printVector to print v4.
    printVector(v4);

    // iterator insert (const_iterator position, size_type n, const value_type&
      val);
    // Use the function insert to insert three 0s at the end of v4.
    // (Notice that the insert function returns an iterator,
    //   but if we do not intend to use it, we can ignore it.)
    v4.insert(v4.end(), 3, 0);

    // Call the function printVector to print v4.
    printVector(v4);

    // bool empty() const noexcept;
    // const_reference back() const;
    // void pop_back();
    // Use a WHILE loop to remove and output each element of v2 backwards.
    // Use function empty for the loop condition, function back to output
    // the last element, and function pop_back to remove elements.
    // (Notice that the insert function returns an iterator,
    //   but if we do not intend to use it, we can ignore it.)
    while (!v2.empty()) {
        cout << v2.back() << " ";
        v2.pop_back();
    }
    cout << endl;

    // void resize (size_type n, const value_type& val);
    // Use function resize to insert three times number 4 in v2.
    v2.resize(3, 4);

    // Call the function printVector to print v2.
    printVector(v2);

    // const_reference front() const;
    // Use function front to output the first element in v4.
    // (Notice that the front function returns a reference.)
    cout << v4.front() << endl;

    // void swap (vector& x);
    // Use function swap to swap v2 with v4.
    v2.swap(v4);

    // Call the function printVector to print v2.
    printVector(v2);

    // Create a new vector v5;
    vector<int> v5;

    // Use the overloaded assignment operator to copy all the elements of v2
    // into v5.
    v5 = v2;

    // void resize (size_type n);
    // size_type size() const noexcept;
    // Delete the last element of v5 by using the functions resize and size
    v5.resize(v5.size() - 1);

    // Call the function printVector to print v5.
    printVector(v5);

    // Create an iterator iterVector5 to point to the first element of v5.
    vector<int>::iterator iterVector5 = v5.begin();

    // iterator erase (const_iterator first, const_iterator last);
    // size_type size() const noexcept;
    // Call the function erase to delete the second half of v5.
    // Use function size to get the range.
    // (Notice that the insert function returns an iterator,
    //   but if we do not intend to use it, we can ignore it.)
    v5.erase(iterVector5 + v5.size() / 2 , iterVector5 + v5.size());

    // Call the function printVector to print v5 again.
    printVector(v5);

    // iterator erase (const_iterator position);
    // Call the function erase to delete the first element of the vector.
    // (Notice that the insert function returns an iterator,
    //   but if we do not intend to use it, we can ignore it.)
    v5.erase(iterVector5, iterVector5 + 1);

    // Call the function printVector to print v5 again.
    printVector(v5);

    // Create a vector of integers named v6 containing numbers from 100 to 105.
    // Using the copy constructor, create a vector named v7, a copy of v6.
    vector<int> v6 = { 100, 101, 102, 103, 104, 105 };
    vector<int> v7 = v6;

    // iterator erase (const_iterator position);
    // iterator insert (const_iterator position, const value_type& val);
    // Erase element 103 from v7 and insert element 333 in its plase,
    // by using an iterator.
    // Note that the function erase returns an iterator that can be used
    // to insert 333 in the right position.
    vector<int>::iterator iterV7 = v7.begin();
    v7.insert(v7.erase(iterV7 + 3), 333);

    // Using a range-based FOR loop, print v7.
    for (auto i : v7) {
        cout << i << " ";
    }
    cout << endl;

    /**************************************************************************
            LISTS
    **************************************************************************/

    cout << "\n\n----------------------------------------------------";
    cout << "\n   ***  STL LIST CLASS  ***  \n\n\n";

    // Use the default constructor to create three lists of integers, intLis1,
    // intList2, and intList3.
    list<int> intList1, intList2, intList3;


    // void push_back (const value_type& val);
    // Use the function push_back to insert the following values in the first
      list:
    // 23 58 58 58 36 15 15 93 98 58
    intList1.push_back(23);
    intList1.push_back(58);
    intList1.push_back(58);
    intList1.push_back(58);
    intList1.push_back(36);
    intList1.push_back(15);
    intList1.push_back(15);
    intList1.push_back(93);
    intList1.push_back(98);
    intList1.push_back(58);

    // Call function printList to print intList1.
    printList(intList1);

    // Using the overloaded assignment operator, copy elements of intList1 and 
      intList2.
    intList2 = intList1;

    // Call function printList to print intList2.
    printList(intList2);

    // void unique();
    // Using function unique, remove all consecutive duplicates in the first 
      list.
    intList1.unique();

    // Call function printList to print intList1.
    printList(intList1);

    // void sort();
    // Using function sort, sort all elements in the second list.
    // (Notice that the function sort can be used only if there are no 
      duplicates.)
    intList2.sort();

    // Call function printList to print intList2.
    printList(intList2);

    // void push_back (const value_type& val);
    //Insert the following elements in the third list:
    //  13 23 25 136 198
    intList3.push_back(13);
    intList3.push_back(23);
    intList3.push_back(25);
    intList3.push_back(136);
    intList3.push_back(198);

    // Call function printList to print intList3.
    printList(intList3);

    // void merge (list& x);
    // Add to the second list all elements of the third list(browse the
    //  list of functions in cplusplus.com to figure out which function
    //  you need to use).
    // --> This is ONE statement only.
    intList2.merge(intList3);

    // Call function printList to print intList2.
    printList(intList2);


    /*************************************************************************
    *       Create statements using the functions below.
    *       Is the output what you expected?
    *************************************************************************/

    cout << "\n(The next output section is determined by your implementation.)\n
      \n";

    // void assign (size_type n, const value_type& val);

    // void assign (InputIterator first, InputIterator last);

    // const_reference back() const;
    // (Notice that this back function returns a reference.)

    // void clear() noexcept;

    // bool empty() const noexcept;

    // const_reference front() const;

    // iterator insert (const_iterator position, const value_type& val);
    // (Notice that the insert function returns an iterator.)

    // void pop_back();

    // void pop_front();

    // void push_front (const value_type& val);

    // void remove (const value_type& val);

    // void reverse() noexcept;

    // void splice (const_iterator position, list& x);

    // void splice (const_iterator position, list& x, const_iterator i);

    // void splice (const_iterator position, list& x, const_iterator first,
      const_iterator last);

    // void swap (list& x);


    cout << "\n\n-----------------------------------------------------";

    cout << endl;
    system("Pause");
    return 0;
}

// Definition function printVector
void printVector(const vector<int> &v) {
    vector<int>::const_iterator iter = v.cbegin();
    vector<int>::const_iterator iterEnd = v.cend();

    for (iter; iter != iterEnd; ++iter) {
        cout << *iter << " ";
    }
    cout << endl;
}

// Definition function printList
void printList(const list<int> &aList) {
    list<int>::const_iterator iter = aList.cbegin();
    list<int>::const_iterator iterEnd = aList.cend();
    while (iter != iterEnd) {
        cout << *iter << " ";
        ++iter;
    }
    cout << endl;
}
```