C++ `list` object

Reference: https://www.cplusplus.com/reference/list/list/

Lists are:

- Dynamically growing or shrinking
- Chained List of values
- Bi-directional sequential scan
- Efficient for insertion and removal operations.

Advantage over Vectors:

- Efficient for insertion and removal operations.

Disadvantages with respect to vectors.

- Cannot have random access.

Constructors
`list();` Creates empty list object with no elements. Example: `list<double> list1;`

`list(size_type n, const value_type& val = value_type());`
Creates a list of specified size and initialized second parameter.
Note that second parameter is optional.
Examples:
`vector<double> list2(10);`
`//size 10, initialized with 0`
`vector<double> list3(5, 100);`
`//size 5, initialized with 100`

`list(const list& x);`
Copy Constructor. Creates new list, copy of parameter list
Examples:
`vector<double> list4( list1 );`
`vector<double> list5 = list2;`

Accessor Methods
`size_type size();` Returns size of list. Examples: `cout << list1.size() << endl;`

`bool empty();`
Returns whether the list is empty (i.e., whether its length is 0).
Example: `if ( !list1.empty() ) { //do something }`

Mutator Methods
`void push_back(const value_type& val);` Adds a new element at the end of the list. The content of val is copied to the new element. Example: `list1.push_back(11.83);`

`void pop_back();`

Removes the last element from the list.

Example: `list1.pop_back();`

`void clear();`

Removes all elements from the list object; list size becomes 0.

`operator=`

Assignment. RHS list is assigned to left side list. Size, data everything changes as per the assigned one.

list2 = list1;

Methods returning references to internal fields (and hence data)

`reference front();`

Examples:

```
cout << "First element: " << list1.front() << endl;
```

Note front() returns reference here, therefore following will do what you can guess? `list1.front() = 11.11;`

`reference back()`

Examples:

```
cout << "Last element: " << list1.back() << endl;
```

Note front() returns reference here, therefore following will do what you can guess? `list1.back() = 9.9;`

Iteration

`iterator begin();`

Returns an iterator pointing to the first element in the list.

`iterator end();`

Returns an iterator referring to the past-the-end element in the list.

Example:

```
for (vector<double>::iterator it = list1.begin();
        it != list1.end(); ++it)
    cout << ' ' << *it;
```

`iterator rbegin(); //reverse beginning`
`iterator rend(); //reverse end`

`iterator erase(iterator position);`
`iterator erase(iterator first, iterator last);`

Removes from the list either a single element (position) or a range of elements ([first, last)).

This reduces the container size by the number of elements removed.

`iterator insert(iterator position, const value_type& val);`

The list is extended by inserting new elements before the element at the specified position. Returns an iterator that points to the newly inserted element.

Examples:

```
list1.insert( list1.begin()+2, 56.99);
```

Relational Operators

`==, !=, <, <=, >, >=`

LHS and RHS are list objects.

For **equality**, comparison goes as following: if size is same, actual

elements are compared sequentially; stops when mismatch is found. For others comparison is done sequentially and stops wherever it is decisive.