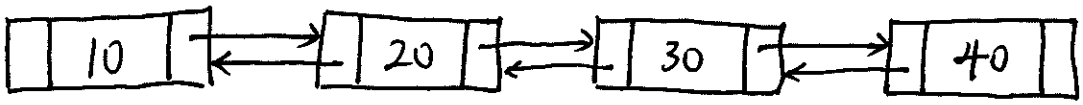
Container: List

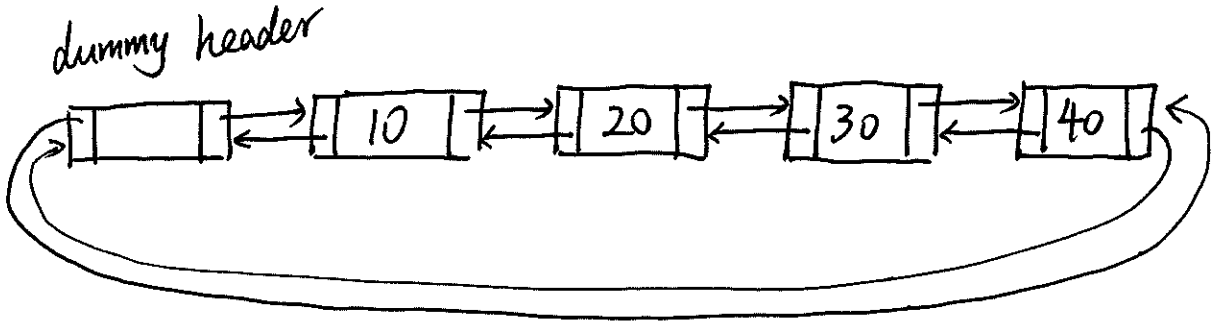
Review:

A doubly linked list is a collection of nodes. Each node contains three fields:

1. an information field holds the actual element on the list
2. a next address field contains the address of the next node
3. a previous address field contains the address of the previous node



List containers are implemented as doubly-linked lists but with a "dummy" header.



Advantages: No special cases for insertion & deletion

end iterator is pointing to "dummy" header

* Lists are sequence containers that allow constant time insert and erase operations anywhere within the sequence, and iteration in both directions (bidirectional iterator).

(Does a vector allow constant time insert and erase anywhere?)
(No. Try insert in the middle of a vector.)

Varieties of Iterators

forward iterator	forward moving (++)	forward_list
bidirectional iterator	forward & backward moving (++) (--)	list, set, map
random access iterator	random access	vector, ordinary pointer

* A random access iterator provides both increment (++) and decrement (--) just like bidirectional iterators, and also provides constant time methods for moving forward and backword in arbitrary size (+7, -4)

e.g. vector iterator can do ++, --, +7, -4

list iterator can only do ++, --

Member functions

- Iterators

  begin() : returns begin iterator (the next of dummy header)

  end() : returns end iterator (dummy header)

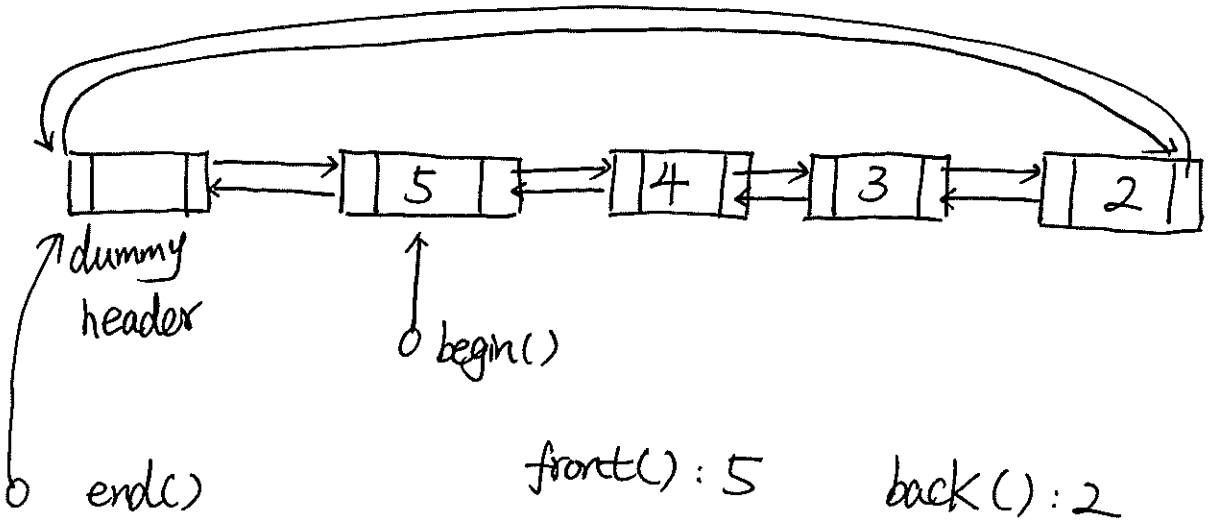
- capacity

  size() empty() max_size()

- Element Access

  front() : access the first element

  back() : access the last element



- Modifiers

  push_back() pop_back() push_front() pop_front()

e.g.

```
#include <iostream>
#include <list>

int main() {
    std::list<int> mylist;
    mylist.push_back (10);          // 10
    while (mylist.back () != 0 ) {
        mylist.push_back (mylist.back() -1 );
    }

    for(std::list<int>:: iterator it =mylist.begin() ; it != mylist.end();
                                                        ++ it)
        std:: cout << ' ' << *it ;

}
```

output : ?

insert()

insert new elements before the element at the specified position

1. iterator insert(const_iterator position, const value_type& val);

   // insert a single element val

2. iterator insert(const_iterator position, size_type n, const value_type& val);

   // insert n elements of value val.

3. iterator insert(const_iterator position, iterator begin, iterator end);

   // insert a range of elements

> return an iterator pointing to the first newly added element.

e.g.

```
std::list<int> mylist;
std::list<int>::iterator it;

for(int i=1; i<=5; ++i) mylist.push_back(i); // 1 2 3 4 5

it = mylist.begin();

++it;                                  // 1 2 3 4 5
                                          ^

mylist.insert(it, 10);                 // 1 10 2 3 4 5
                                             ^

mylist.insert(it, 2, 20);              // 1 10 20 20 2 3 4 5
                                                 ^

--it;                                  //          ^

std::vector<int> myvec(2, 30);

mylist.insert(it, myvec.begin(), myvec.end()); // ?
```

erase()

remove either a single element or a range

1. iterator erase(const_iterator position);

2. iterator erase(const_iterator begin, const_iterator end);

return an iterator pointing to the element that followed the last erased element

e.g.

```
std::list<int> mylist;
std::list<int>::iterator it1, it2;
for(int i=1; i<10; ++i) mylist.push_back(i*10);
                    // 10  20  30  40  50  60  70  80  90
it1 = it2 = mylist.begin();
advance(it2, 6);                                    ↑
++it1;                  ↑
it1 = mylist.erase(it1); // 10  30  40  50  60  70  80  90
                            ↑               ↑   ●
it2 = mylist.erase(it2); // 10  30  40  50  60  80  90
                            ↑               ↑
```