20. Containers & Iterators

Storing data
- access via array
- vector< >
- list
- etc.

Iteration

array: `for (i=0; i<n; ++i) f(a[i])`
`for (p=a; p<a+n; ++p) f(*p)`

linkedlist: `for (p=head; p≠null; p=p->link) f(p->data)`

vector: `for(i=0; i<v.size; ++i) f(v[i])`
`for(p=v.begin; p≠v.end; ++p) f(*p)` ✓

notice: int index only works with direct access data struct, not linear struct

find(key): what to return for not found
find (highest) – what return if container empty

out of band value? null? – not for nonptr.
out of band index? (-1)
pointer

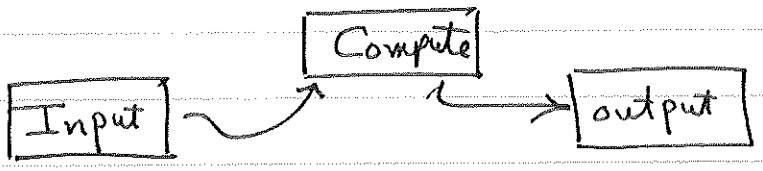
string::find_first_of ⟶ string::npos
= (size_t)(-1)
which is largest possible pos value

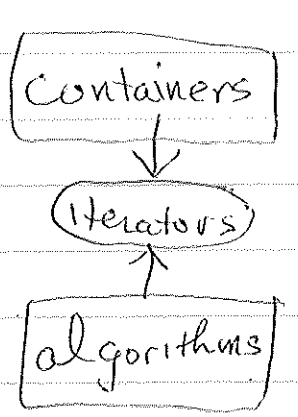
Iterators have out of band value built in
Java generics – can only return ptr ∴ null
C++ return v.end() for not found.

Sort recursively
i = v.begin()
e = v.end()
m = i + v.size()/2 ← only if direct access iterator.

Sort(i, m)
sort(m, e)
merge

20.2 STL Ideals





algorithms :- sorting
- searching
- join with computation

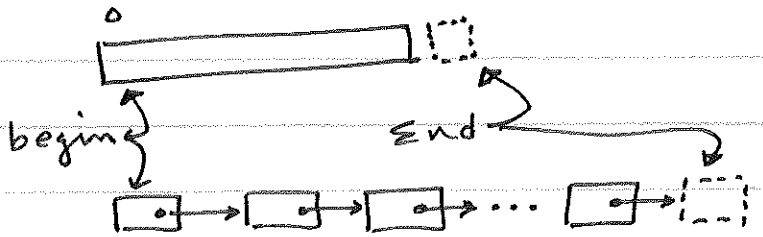
What do we do with data?
- collect
- organize for access
- retrieve by index, value, condition
- modify : add, remove, sort
- algorithms can ignore data properties
  eg. int vs double

vector<int>
vector<string> } same to algs & data structs

We want

- uniform access independent of how stored type
- type safe access
- easy traversal
- compact storage
- fast: retrieval, addition, deletion
- standard algorithms
  - copy, find, search, sort, sum, ...

20.3 Sequences & Iterators



for a list end is: NULL
- a sentinel node

sentinel better if bidirectional itor

for direct access:

invariant: end - begin ≡ size
begin + size = end

- no off by one problem
- zero is a better 1st index than 1

Iterator basic opns

$p \to f \equiv (*p).f$

`p == Q`
`p != q`

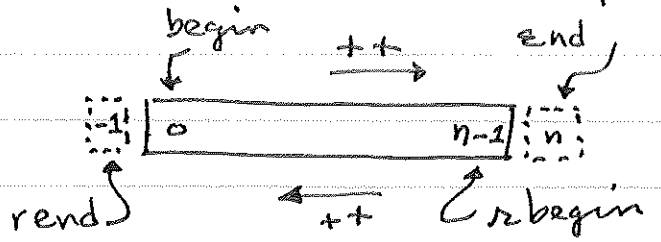
`*p = xc` (non const)
`xc = *p` (const)

`p -> f = x`
`x = p -> f`

ref `++p`
value `p.++`

(for any fwd iterator)

note: reverse_iterator is a forward iterator.



range [begin: end) in math notation

connect algorithms to data

algorithms: sort, find, search, copy, etc.

ITERATORS

data: vector, list, map, array, deque, etc.

```
// Find largest value
// Return itor to value, end if none
template <typename itor>
itor high (itor begin, itor end) {
        itor high = begin;
        for(itor p = begin; p != end; ++p){
                if(*high < *p) high = p;
        }
        return high
}
// only assume ∃ operator<
```

opt arg less

20.4 Linked Lists

```
template <typename elem>
struct list;
struct link {
        elem val;
        link *prev;
        link *next;
};
```

```
template <typename elem>
struct list {
        link<elem> *head;
        link<elem> *tail;
        class link;
};
```

list operations
- as for vector, but no oper[]
- insert, erase arbitrary positions
- iterator

~~20.3~~ 20.4 list operations

Ch. 20 6

Iterator is a class member

```
template <typename elem> {
    class iterator;
    iterator begin();
    iterator end();
    iterator insert(iterator p, const elem &)
    iterator erase(iterator p);
```

also push_back pop_back back
push_front pop_front front

```
template <typename elem>
class list <elem> ::iterator
    node <elem> *curr;
    iterator & operator ++() {cur=cur->next; ret *this
    iterator & operator --() {cur = curr->prev; ret *this
    elem & operator*() {ret cur->val}
    bool operator == (const iterator &b) {ret cur
                                          ==b.cur
    bool operator != (const iterator &)
```

begin() == end() → empty seq
∴ not a special case

~~20.6 text editor~~

BOOST

ch. 20
7

20.6 text editor

text stream → data struct → text stream
insert/delete lines/chars
search
display → screen

vector<char> → BAD because
long insert delete

list<line> where line is vector<char>

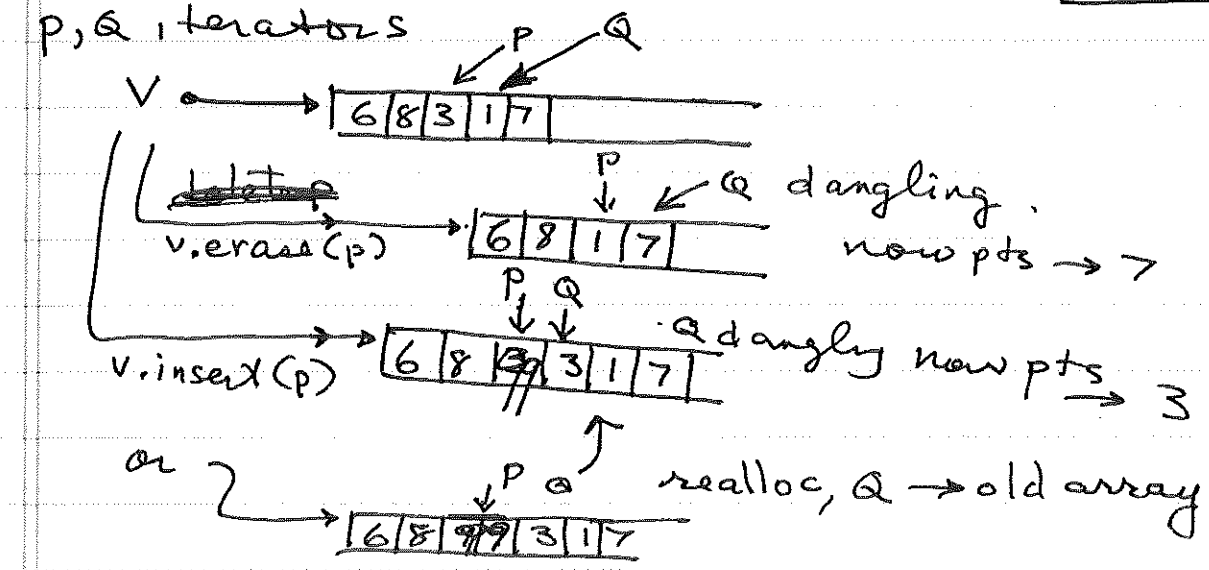
or: list<string> vector<wchar_t>
document inherently 2-dimensional
editing: ASCII | ISO-Latin-1, ..., 15
Unicode as UTF8, UTF16, wchar_t
Σ 1bitcode

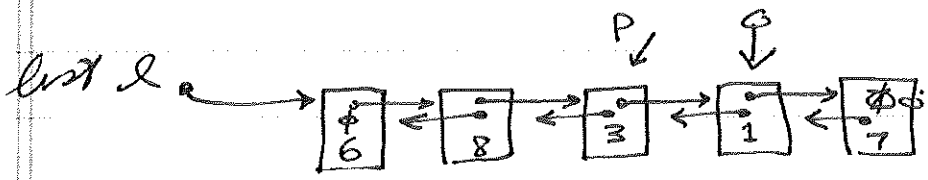
why list? – flex insert/delete
<line> but goto line is slow
↑↓ fast Ø, $ fast

a line: char[] — no size, no begin/end, can't pass
vector<char> } reasonable
string – search easier
list<char> – complete node & char
lots of memory

20.7 insert / erase

p, Q iterators





delete p sets p == Q
insert p new elt between p & Q
Q not dangling

list: insert, erase O(1) | BUT op[] is O(n)
vector: insert, erase O(n)
↳ push_back, pop_back
sometime O(n) but amortizes to O(1)

storage: O(n) both, but
vector: fixed overhead
list: overhead fixed
+ O(n) ptrs per node

20.10 Containers

vector
list
deque - cross list, vect
map - balanced BST
multimap - BST, non-unique keys
unordered_map - hash table
- need good hash fn
- iterator order not predictable

unordered_multimap
set, multiset, unordered_set, unordered_multiset
array - [fixed size] - ex: numerics

"almost" containers
T[n] - chunk of storage a[i] ≡ *(a+i)
string - char only, but many char traits
valarray - numeric apps

20.10.1 Iterators

input : ++, read (x = *i), ==, !=
output : ++, write (*i = x), ==, !=
forward : input ∪ output
both rd, wr unless const
(*p).m ≡ p→m
bidirectional : also op --
randomaccess : elt access via *i and i[j]
also i+j, i-j, i+=j, i-=j
< <= > >=