cse 330

1-9-12 *errata list update in book

Documentation

In each file: /** . . . . . *

name

date, started and stopped.

filename

Problem description

Algorithm used and data structure used.

*** . . . . . */

Each function: /** . . . . *

Short description.

** * . . . . */

indentation format

4 spaces for indent (Dont use tabs)

examples.

θ space

```
if θ ( i θ< θ10 ) θ {          for θ(i=0;θ i <10;θi++) {
  θ θ θ θ                        θ θ θ θ . . . .
  θ θ θ θ
} else {                       }

}
```

end of functions

```
main()
{
} //main
```

- user defined class names start with uppercase letters.
- objects should start with lower case letters.
- Do not abbreviate names!

include files

test.h

```
#ifndef TEST_H
#define TEST_H
   =
   =
   =
   =
#endif
```

now test.h can be included without causing multiple definition error

main.cpp

```
#include "test.h"
main()
{   void f();
    f();
}//main
```

f.cpp

```
#include <iostream>
#include "test.h"
f()
{ cout << "f()\n";
} //f
```

make files

```
make -f main.make
```

```
main: main.o f.o
    tab -> g++ -o main main.o f.o
main.o: main.cpp test.h
    tab -> g++ -c main.cpp
f.o: f.cpp
    tab -> g++ -c f.cpp
```

main.cpp

```
int i;
main()
```

f.cpp

```
extern int i;
f()
```

- extern is an example of a Storge class

extern
automatic
static
register

Lab notes - infix postfix Lab1

a+b*c = a + (b*c)

ab+

- Stack is an adapter

a*b+c = ab*c+

a+b*c = abc*+

(a+b) * (c-d) = ab+cd-*

((a+b))/c) * d = ab+c/d*

a+((b/c)*d) = abc/d*+

v.back = v[v.size()-1]

```
vector<int> v;
v.push_back(7);
v.push_back(10);
```

v.back - top of stack

v.front - bottom of stack

v.empty - Boolean, true if empty.

Stack in STL

```
#include <stack>
stack<int> first;
stack<char> second;
```

- input one character at a time.
- type out prec()
- use # for end of expression.

```
if (ch >= 'a' and ch <= 'z')
```

1-11-12

Homework 1 : Ex 8 pg 21

Variance

main()
{ input until eof
    call variance
}

double variance (array input)
{
  return variance.
}

Sample run

$ script

^ D control d

puts everything in a file called typescript

Ascii codes

char	dec	hex	binary
0	48	30	00110000
1	49	31	
9	57	39	
A	65	41	01000001
Z	90	5A	
a	97	61	
z	122	7A	

```
#include <iostream>
#include <cstdio>
using namespace std;
main()
{   char ch = 'a';
    int i;
    i = ch;
    cout << ch << " " << i << endl;
    printf ("%c  %d"),ch,ch);
}
```

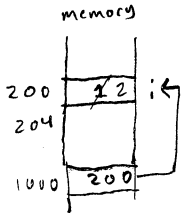
output = a 97
a 97

- dereferance means content
- scaling p=a+1 = 1004

Pointers, Arrays, strings, Parameters

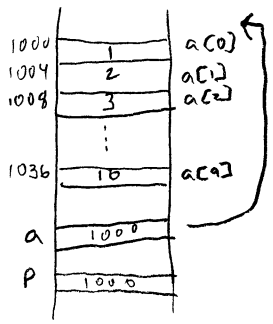
- pointers are always 4 bytes
- array is basically a pointer

```
main()
{ int i = 1, *p;
  p = &i;
  *p = 2;
  cout << i; → 2
```



array names are (constant) pointers

```
main()
{ int a[10], *p;
  p = a;
  *p = 1;
```




```
for (p=a+1; p<a+10; a++)
{   *p = *(p-1) + 1;
}
for (p=a; p<a+10; p++)
{   cout << *p << endl;
}
```

pass by value.

pass by reference

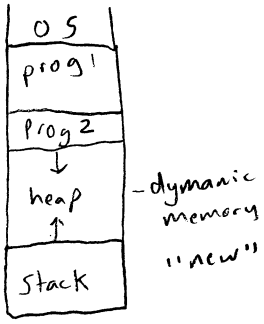
Parameters

```
f(int x)
{ x=3
}

main()
{ int i=2
  f(i);
  cout << i;
}
```

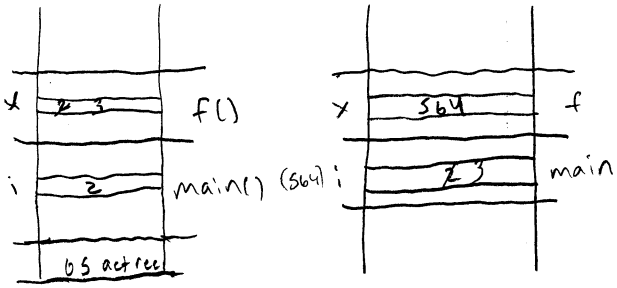
```
f(int * x)
{ *x=3
}

main()
{ int i=2
  f(&i);
  cout << i;
}
```



-dymanic memory "new"

-default return type is int.



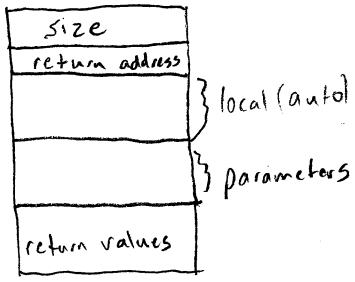
Stack of activation records.

c++ version of above

```
f(int &x)
{ x=3
}

main()
{ int i=2
  f(i);
}
```

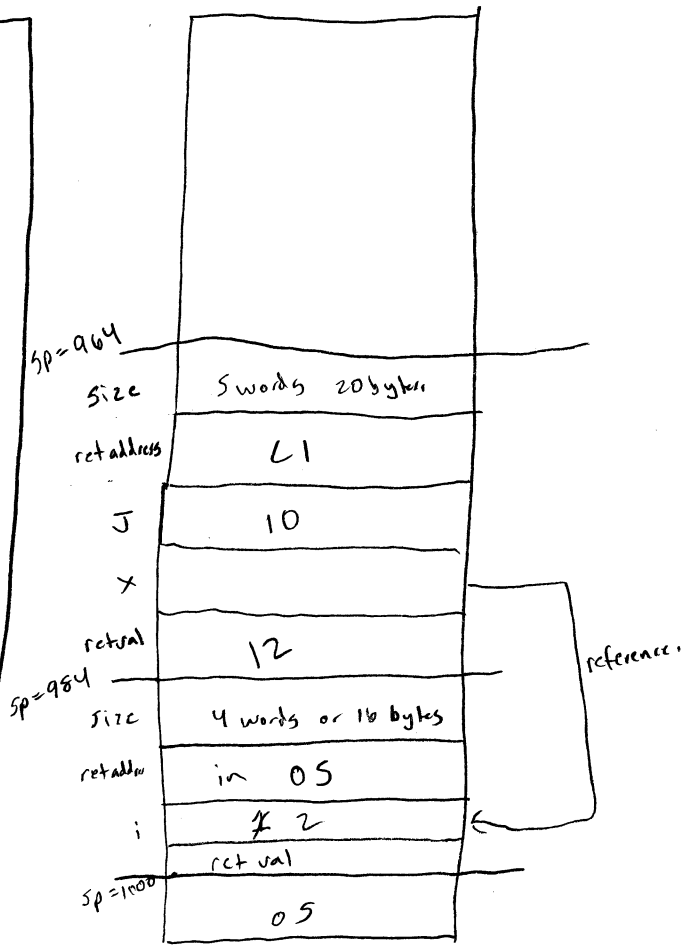
template of any activation record



Prog.cpp

```
int y
int f(int &x)
{  int j=10
   x++
   return x+j
}

int main()
{  int i = 1
   L1: y= f(i)
   L2: cout << i << y
   L3: f(i)
   L4: cout << i << y;
   L5: return 0
}
```



2\2

$ a.out

$ echo $? looks at return value of main program.

1-18-12 In lab report, typescript or screen shot
every function, time and storage complexity
email actual program to Avi

Time Complexity

- we are interested in the maximum number of times an algorithm "loops", not how much time it takes.

↓ while loop
calls itself
calls another alg that calls it back

Ex Alg 1

```
input n
sum = 0
for i = 1 to n
    sum += i
output sum
```

#times loop : n , set - all functions related to f(n) = n

time complexity is in O(n)

everything is based on the size of n.

Ex Alg 2

```
input n
Sum = 0
for i = 1 to n
    Sum += i        } work
    output sum
```

loops n times $\Rightarrow O(n)$

Ex Alg 3

```
input n
Sum = 0
for i = 1 to n
    Sum += i
output sum
Sum = 0
for i = 1 to n
    for j = 1 to n
        Sum += j
output sum
```

loops $n + n^2$ times $\Rightarrow$ time complex $O(n^2)$

$O(n^2) = \{ n+n^2, n^2, 5n^2, .0001n^2 + 20z, \ldots \}$

+ we ignore constants and smaller functions.

In ex 1 we have $c_1 n^0 + c_2 n + c_3 \Rightarrow O(n)$

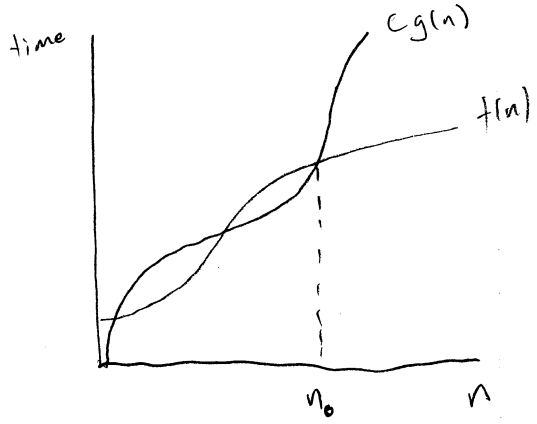
In ex 2 we have $c_1 + c_4 n \Rightarrow O(n)$

In ex 3 we have $c_1 + c_2 n + c_5 + c_6 n^2 + c_3 \Rightarrow O(n^2)$

Definition -

$t(n) \in O(g(n))$ iff $t(n) \leq c\,g(n)$ where $c > 0$ & $n \geq n_0$

$t(n)$: time taken by an algo



Ex Alg1

suppose $c_1 = 4$ $\quad 4 + 2n + 3 = 2n + 7$ μsec

$c_2 = 2$

$c_3 = 3$ $\quad 2n+7 \leq 3n$ for $n \geq 7$

$2n+7 \in O(n)$

n	2n+7	3n
1	9	7 3
2	11	7 6
⋮	⋮	⋮ ⋮
from 7 on → 7	21	= 21
8	23	< 24

Ex Alg4

input n

sum = 0

i = 1

while i < n

sum += i

i = 2 * i

output sum

if $32 = 2^5 <= 7 \quad \log_2 32 = 5$

$16 = 2^4 <= 7 \quad \log_2 16 = 4$

time $O(\log n)$

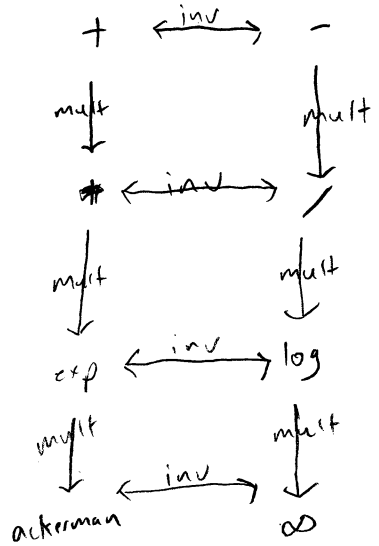
$c_1 + c_2 \log_2 n + c_3 \leq c \log_2 n \Rightarrow O(\log_2 n) \Rightarrow O(\log n)$

function	time	space
main()	O(n)	O(n)
variance()	O(n)	O(n)

inv = inverse
mult = multiple

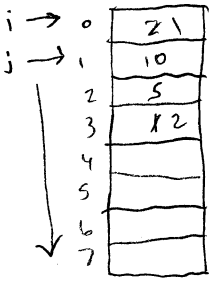
Ex Alg 5	n	i
input n	32	32
sum = 0		16
i = n		8
while i > 1		4
sum += i		2
i = i/2		1
output sum		

time = O(log n)



Selection Sort (Array[0, ... n-1]

```
for i=0 to <n-1
    for j= i+1 to <n
        if a[i] > a[j]
            swap( a[i], a[j])
```



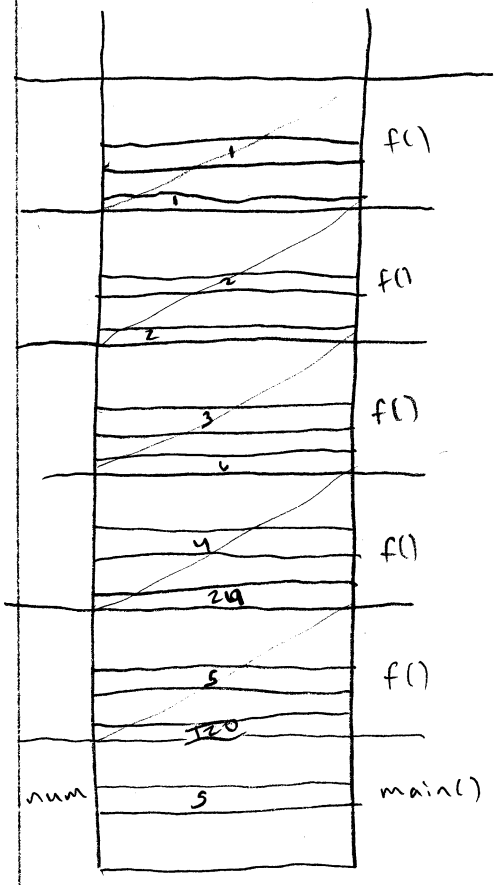
time = $(n-1)+(n-2)+(n-3)+\dots+2+1 = \frac{(n-1)n}{2} \Rightarrow n^2-n \Rightarrow O(n^2)$

$1+2+n = \frac{n(n+1)}{2}$

```
f(int n)
{ if (n <=1) return 1
   return n * f(n-1)
}

main()
{ int num
  cout << "enter num"
  cin >> num
  cout << " fact( " << num << ") = " << f(num)
}
```

Stack of activation records.



time = O(n)

fact(5) = 120

char = 1 byte
int = 4 bytes

1-23-12 # Pointers & Operator overloading

C char strings

null terminated array of characters.

char str[10] = "test"

'\0' = 0 in ascii

'0' = 48 in ascii

'0' ≠ '\0'

0 is false, everything else is true

```
int strlen (char * s)
{ int i
  for(i=0; s[i] ≠ '\0'; i++);
  return i
}
```

null

for(i=0; s[i]; i++ → for(i=0, *s; i++, s++)

↑ comma

for(int i=0; *s++; i++);

↑ post increment

memcpy
strcat
strncpy
strlen
strcpy
} <string.h>
<cstring>

<string>
c++ string

```
void strcpy( char *t, char *s)
{
   for(; *t = *s; t++, s++);
}
```

```
for(; *t++ = *s++; );
```

```
main()
{ char str1[10] = "test"
  char str2[5] = "ab"
```

String a = "abc", b = "1234"
a = a + b

operator overloading

strcpy

strcat

Chapter 7 - String
Chapter 4 - time complexity

Modulo 3 Arithmetic - only numbers 0, 1, 2

0	1	2	
0	1	2	2+1 = 3
3	4	5	|
6	7	8	
9	10	11	
12	13	14	14/3 = 4 rem 2

```
modulo m(5); // 0,1,2,3,4        class Modulo {
modulo m(3)                       private:
m=3 n=8                             int n; // base
                                    int value; //value
things we want to do:               public:
  m=m+1 → 4                           Modulo(); // default constructor
  m++ → 0                             Modulo(int m); // modulo m(5);
  m=m+n → 2                           Modulo operator=(int) // assignment n=8
  m=++n                               Modulo operator+(int) // m=m+1
  m=n++                               int getValue(); // returns value
cout << m << n                        Modulo operator++(); // pre increment
                                      modulo operator++(int); // post increment
                                  };
```

not required
default private

mod value

```
Modulo::Modulo() {
  n=2
  value=0
}

Modulo::Modulo(int m) {
  n=m
  value=0
  assert(n>0)
}
```

#include <cassert>

```
modulo modulo::operator=(int op) {
  value=op;
  value=value%n;
  return *this; // pointer to implicit parameter
}

modulo::modulo(int m, int v){
  n=m
  value=v%m
}
```

Why time complexity,

Lab 2

selection sort

n	time	$c = time / n^2$
1000	.008	.000000008
2000	.032	.000000008
3000	.066	.000000007
4000	.108	.000000007
5000	.160	.000000006
6000	.220	.000000006
7000	.295	.000000006
8000	.370	.000000006
9000	.455	.000000006
10000	.551	.000000006

1-25-12

modulo continued

```
public:
  modulo (const modulo & m)
  modulo operator=(modulo op)
  modulo operator++ () // prefix
  modulo operator++ (int) // prefix
```

```
modulo modulo::operator=(modulo op) {
    value = op.value;
    n = op.n
    value %= n;
    return *this
}
```

Example

m1 = ++m2

```
modulo modulo::operator++() {
    value ++;
    value % n;
    return *this
```

example.

m1 = m2++

```
modulo modulo::operator++(int) {
    modulo temp = *this; // copy of implicit
    ++(*this);
    return temp;
```

outside of class << belongs to ostream

```
ostream & operator<<(ostream & out, Modulo n);
{ out << n.getvalue();
  return out;
}
```

Strings

end points to past end()

In STL:

```
String s = "abc"
String s ("abc");
cout << s.length();
cout << s[1];
s += "de"; // abcde
s.insert (2, "123") // ab123cde
s.replace()
s.find()
```

generic functions

```
count (s.begin(), s.end(), "i")
i = find ( "  "  "  " )
```

String

```
class String {
    int size;
    char * buffer; // allocate memory from heap
public:
    String ()
    String (const char *)
    String (const String &)
    ~String (); // deallocate memory
```

Big 3 copy constructor
destructor
assignment operator strlen need <cstring>?

```
String::String()
{   size=0
    buffer=0
}
```

```
String::String(const char * p)   // String S("abc");
{   size = strlen(p);
    buffer = new char[size];
    for(int i=0; i < size; i++)
    {   buffer[i] = p[i]
    }
}
```

```
String::String(const String & source)
{   size = source.size;
    buffer = new char[size]
    for(int i=0; i < size; i++)
    {   buffer[i] = source.buffer[i];
    }
}
```

→ memcopy(buffer, source.buffer, size)

```
String  String::operator=(const char * p)
{   delete [] buffer;
    size = strlen(p);
    buffer = new char[size]
    for (int i= 0; i < size; i++)
    {   buffer[i] = p[i]
    }
}
```

S 01234

2 56

7 - 2 = 5

5

War - the game

1-30-12

Two players
Deck of cards
Each player 3 cards initially
Each player draws one of her cards
Highest Rank card wins (2 pts)
a draw results in one pt for each player.
players replace their drawn cards from the deck
game ends when player hands are empty (23 rounds)
Books version ends when deck is empty
player with highest score wins.
ace has lowest rank, king has highest

Lets go over bottom up design

nouns are classes
verbs are member functions

enum Suites {diamonds, clubs, hearts, spades}
0 1 2 3

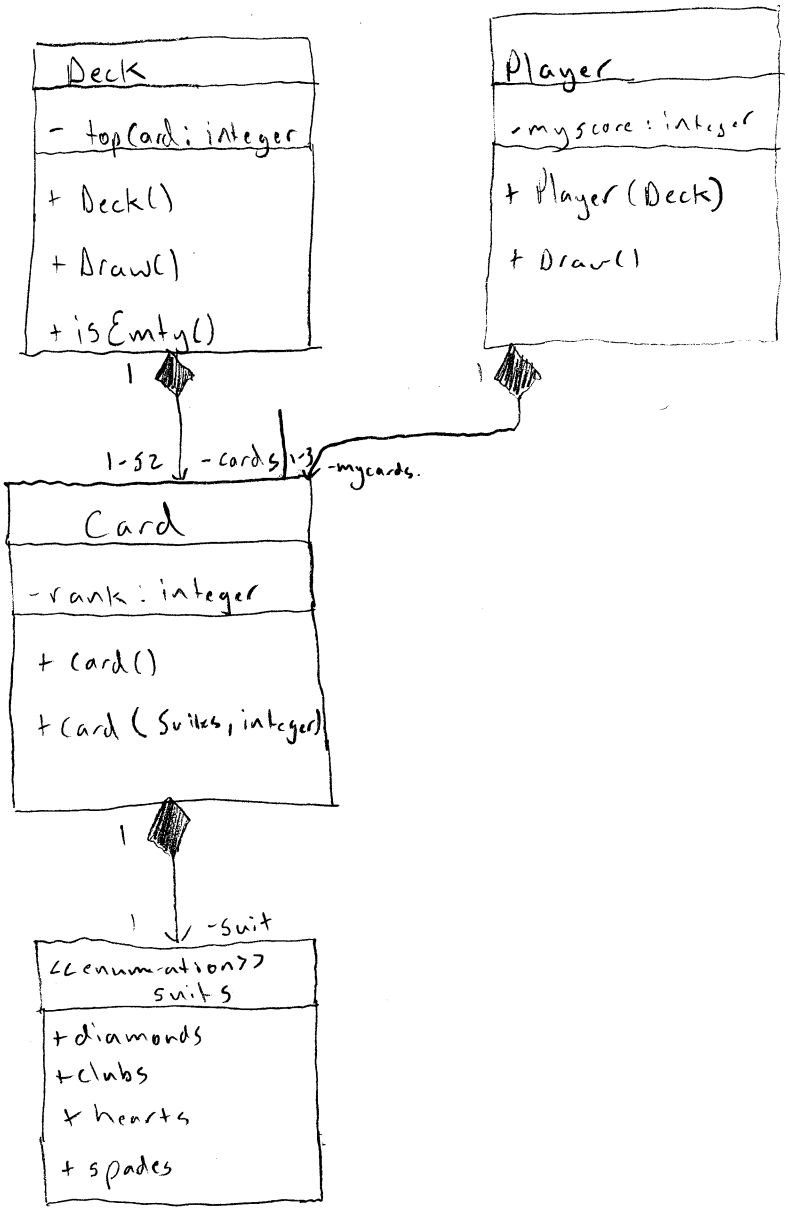
enumeration

```
class Card {
private:
    int rank
    int suit → Suites suit;
public:
    Card()
    Card(suites, int)
};

class Deck {
private:
    Card cards[sz];
    int topCard; //pointer top card
public:
    Deck()
Card Draw()
void shuffle()
Bool isEmpty()
};
```

- means private.
+ means public.

UML



Functors

an instance of random integer is a function object or functor.

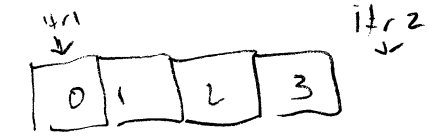
A functor is an object acts like a function (an object that can be called like a function).

A functor overloads a function called operator.

A number of functions in STL expect functors as a parameter.

Ex: random shuffle, for_each

random shuffle



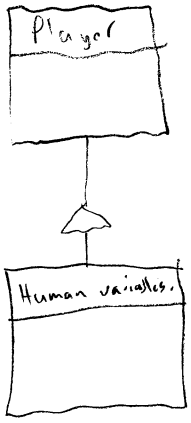
swap (obj 0, obj (0+1)) 0

swap (obj 1, obj (1+1)) 0,1

swap (obj 2, obj (2+1)) 0,1,2

2-1-12

War game continued.



Clean in make file.

Vector Lab

type def = define new types.

T * iterator
new name for T* iterator

```
template <class T>
Vector<T>::Vector() {
    mySize=0
    myCapacity=0
    buffer=0
}
```

```
Vector<T>::Vector(unSize)
my.
 buffer= new T[size];
for(int i=; i<size; i++)
{ buffer[i]=T()
```

↘ called constructor

Operator =

mysize = v.mysize

mycapacity = v.mycapacity

delete [] buffer

buffer = new T[my.capacity]

begin() beginning of buffer

end() buffer + size

:: scope resolution operator

2-6-12 mid term feb 15th

template

swap.h

T= template parameter

```
template < class T >
void swap ( T & x , T & y)
{   T temp = x
    x = y
    y = temp
}
```

$ g++ -c swap.cpp

```
String str ("abc")
String :: iterator it = str.begin()
for( ; it != str.end(); it++)
{   cout << *it;
}
```

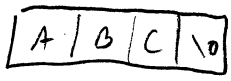
```
class String {
public:
    typedef char * iterator
    iterator begin () { return buffer}
```

reverse order

```
String:: reverse_iterator j
for ( j = str.rbegin(); j != str. rend(); j++)
        cout << *j
```

```
i = str.begin(), i++, i++
for( ; j != str.begin() ; i++)
        cout << *i        prints "cb"
```

str.c_str() → returns char *



friend classes can access everything, private & protected fields

Table 6.1 lookup

Lists - container class

```
list<int> aList
aList.push_back(1)
aList.push_back(2)
aList.push_front(3)
aList.push_back(4)
```

aList → [1] → [2]

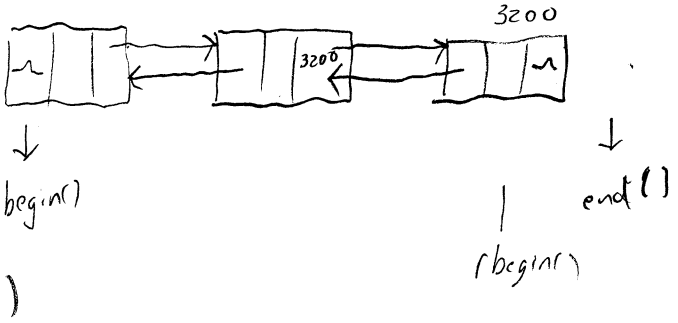
aList → [3] → [1] → [2] → [4]

```
for(list<int>::iterator i = aList.begin() ; i != aList.end() ; i++)
    cout << *i
```

2-8-12

Chapter 9 Lists

elements are linked together using pointers / addresses



use when number of elements is not known in advance and random access is not required.

operator[]

List vs vector

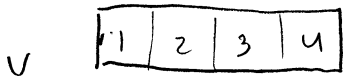
V.front() = V[0]

name	operation	List	Vector
Access	front()	O(1)	O(1)
	back()	O(1)	O(1)
	*iter	O(1)	O(1)
	operator[]	X	O(1)
add new	push_front()	O(1)	X
	push_back()	O(1)	O(n)
	insert(iter, val)	O(1)	O(n)

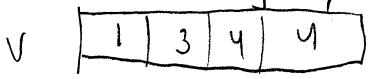
find generic

Binary search requires random access

Remove	pop_front()	O(1)	X
	pop_back	O(1)	O(1)
	erase(iter)	O(1)	O(n) → generic algorithm
	remove(val) -all occurrences.	O(n)	remove(iter, val) O(n)
inclusion test	find(iter, iter, val)	O(n)	O(n)
	binary_search(iter, iter, val)	X	O(log n)



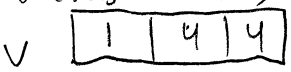
remove(v.begin, v.end, 2)



vector<int>::iterator itr = v.begin

iter++

v.erase(iter)



find returns null pointer if nothing found.

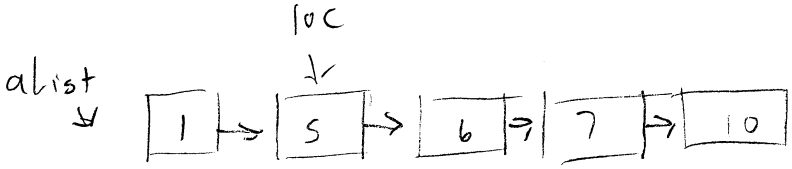
Binary Search

0	10
1	15
2	20
3	25
4	30
5	40
6	50
7	70
8	75
9	76
10	77

↑ sorted

20 is value

compare to middle element of 40. reduce search in half everytime.




```
list<int>::iterator loc = find (alist.begin(), alist.end(), 5)
loc = alist.insert (loc, 10)
```

→ [10] → [5] →
loc

```
list<int>::iterator stop = find ( loc, alist.end, 7)
```

10 → 5 → 6 → 7
loc stop

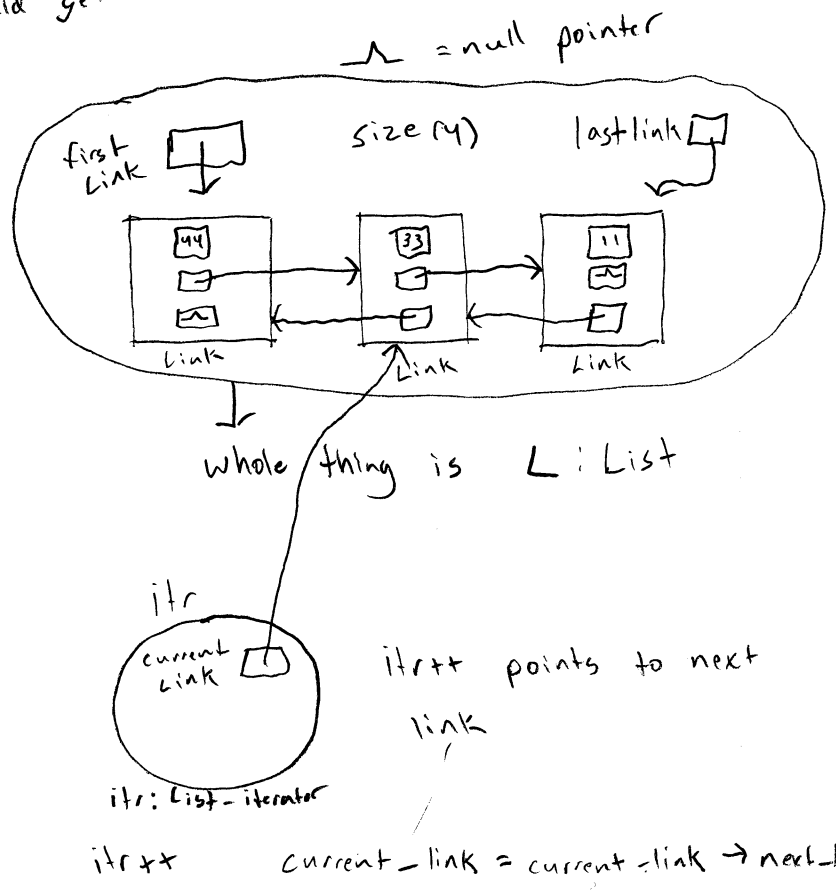
```
alist.erase ( loc , stop)
```

alist 1 → 7 → 10

```
num = count (alist.begin , alist.end() ,10)
```

↓
1

*itr should get value




```
List<int> L;
L.push_back(44)
"          33
"          11
"          22

List<int>::iterator itr
itr = L.begin()
++itr
cout * itr
```

*

```
template<class T>
class Link {
    T value
    Link * next_link
    Link * previous_Link
friend class List(T)
  "      "  List_iterator(T)
};
```

```
template <class T>
class List {
    int size;
    Link * first_link
    Link * last_link
public:
    push_back(T)
    pop_back()
    push_front(T)
    pop_front()
    insert(iterator, T)
};
```

```
template <class T>
class List_iterator {
    Link * current_link
public:
    T operator * ();
    List_iterator operator++()      // prefix
    "  "  "  "  "  "  "  "  (int)   // postfix
friend class List<T>
```

Link Constructor

```
value = x
next_link = previous_link = 0
```

constructor for List

```
size = 0
first_link = last_link = 0
```

List Pushback

```
List :: push_back (x)
    new_link = new Link (x);
  if (first_link == 0) {
    first_link = last_link = new_link
  }
  else {
    newlink -> previous link = last_lank
    lastlink -> next link = new-link
    lastlink = newlink
  }
size++
```

List begin

```
List::begin
    return List_iterator (first_link)
```

List iterator *

```
operator*
return current_link -> value.
```

```
return * currentlink .value.
```

erase

two iterators

p → next_link = n

n → prev_link = p

delete iter; current_link

insert

insert befor iterator

new Link

two iterators

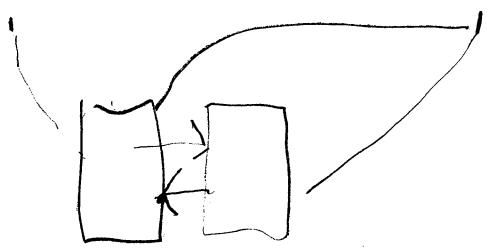
2-13-12

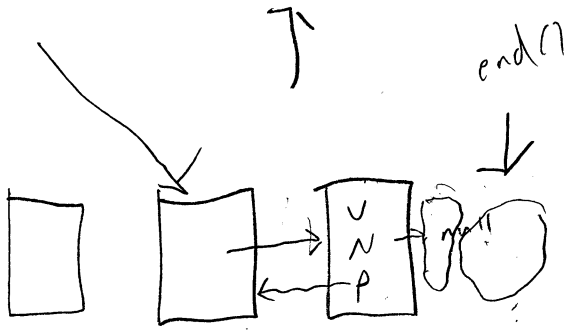
Binary Search

must be sorted

T(n) = time for bin search rec with n elements. = high - low

$T(n) = T(n/2) + C$





m = itr++

m = ++itr

Stack & Queue Containers

2-20-12

Stack		Queue	
push	push_back	push	push_front
pop	pop_back	pop	pop_back
top	back	top	back
empty	empty	empty	
size	size	size	

Container Header

template < class T, template <class T > class Container >

Allocator manages memory efficiently

2-22-12

- Iterator traits can tell you the type of object you are pointing to.

why $n \cdot \log n^2$

n elements

$T(n)$ = time to sort n elements

$T(n) = T(n/2) + T(n/2) + cn$ — everytime size divides by 2

recurrance

$$T(n) = 2T(n/2) + cn$$
$$= 2\left(2T(\tfrac{n}{4}) + cn/2\right) + cn$$
$$= 2^2 T(n/4) + cn + cn$$
$$= 2^2 T(n/4) + 2cn$$
$$= 2^2\left(2T(n/8) + c\tfrac{n}{4}\right) + 2cn$$
$$= 2^3 T(n/8) + cn + 2cn$$
$$= 2^3 T(\tfrac{n}{8}) + 3cn$$

Assume $n = 8$, $\log_2 n = 3$

$$n \times T(1) + \log_2 n(cn)$$
$$cn + cn\log n$$
$$n\log n$$

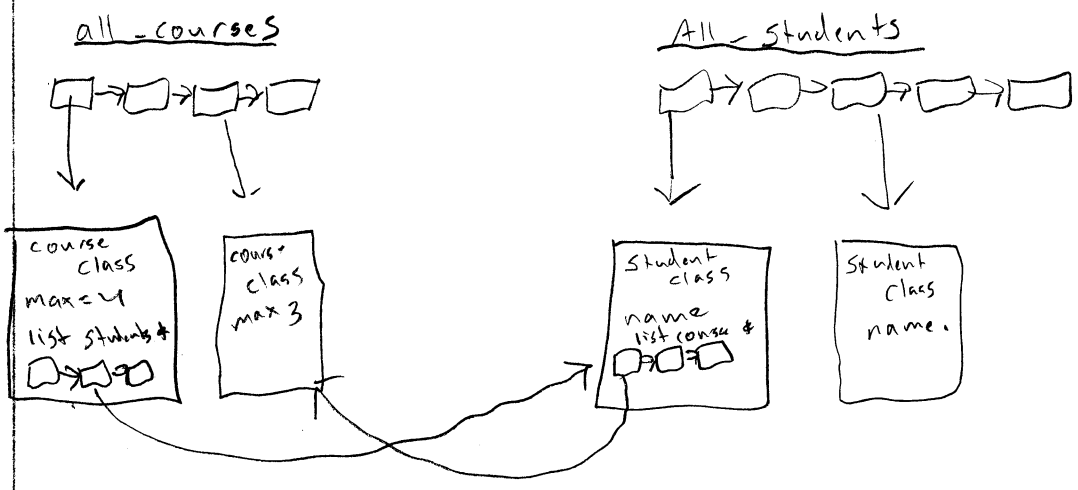
generic list

Homework 3 - course registration system

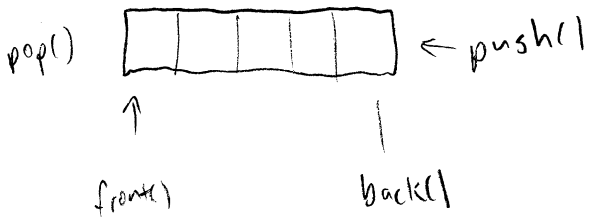
global / in main() lists:

list <course *> all-courses

list < student *> all - students



Queue




```
#include <queue>
queue<int> q;
q.push(1)
q.push(2)
cout << queue.front() // 1
```

Deque - double ended queue

- one of most versitile data structures

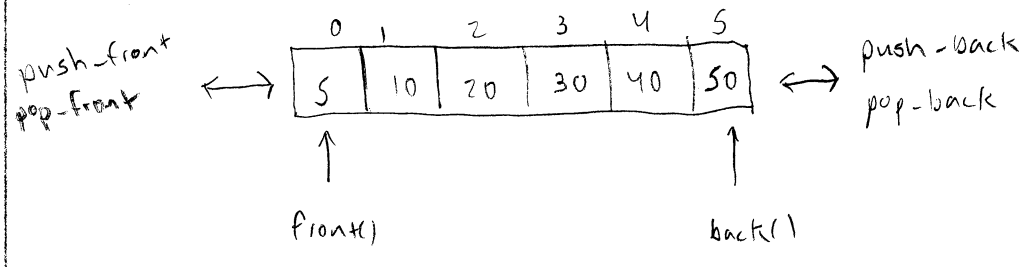
Example: Deck of cards

STL defines it in #include <queue>

similar to queue but has [] operator

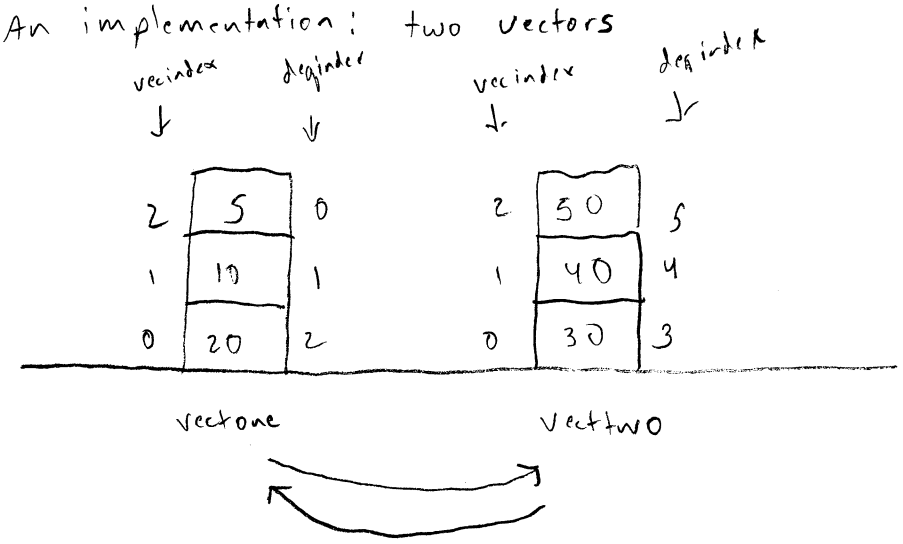
↓

takes O(logn)




```
deck<int> d
d.push_back(30)
d.push_back(40)
d.push_front(20)
d.push_front(10)
d.push_front(5)
d.push_back(50)
```

An implementation: two vectors



back() into deque = back() vec two

front() into deque = back() vec one

```
template <class T>
T deque::front() {
    if(vecOne.empty()
        return vecTwo.front()
    else
        return vecOne.back()
```

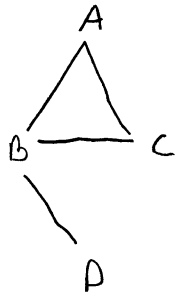
```
T& deque::operator[](int index) {
    int n= vecOne.size()
    if (index < n)
        return vecOne[n-1-index]
    else
        return vecTwo[index-n]
}
```

2-27-12 Trees ; sets - Chapter 12 ; 13

Trees

- a tree is an acyclic graph
- a graph consists of a set of nodes and set of edges.
- an edge connects a pair of nodes

Ex. G_1



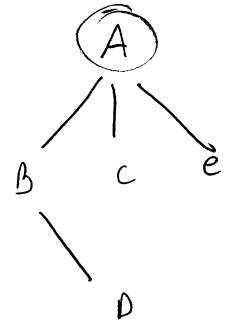
G_2

G_1: $\langle N, E \rangle$

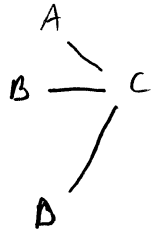
$N = \langle A, B, C, D \rangle$

$C = \{(A,B), (A,C), (B,C), (B,D)\}$

T_1:



T_2:



- In a tree, one of the nodes is designated as the root.
- root can be any one of the nodes.

- in a binary tree, no node has more than two children

- longest path from root to farthest leaf is height

<stmt> → <select stmt> | <expr> | <while> | <comp stmt>
<comp stmt> → { <stmt>; <stmt>* }
<assign> → identifier = <expr>

- <u>complete binary trees</u>, everything pushed up and to the left.

$h = \lfloor \log_2 n \rfloor$ $h=2$ $4 \leq n < 8$
$h=3$ $8 \leq n < 16$
$h=4$ $16 \leq n < 32$

truncates fraction portion
3.27 → 3.00

- <u>height balanced binary tree</u> - height of left subtree and right subtree doesn't differ by more than 1.

- <u>binary search tree</u> - complete or height balanced

- root is only access point for a tree

- In a complete binary tree, all nodes are pushed up and then pushed left
- a complete binary tree of height h has between 2^h and $2^{h+1}-1$ nodes

Or

- a complete binary tree containing n nodes has height of $\lfloor \log_2 n \rfloor$

floor of

- in a height balanced binary tree at every node, the difference between the right and left subtrees is not larger than one.

- a complete binary tree is height balanced and this implies that the largest number of nodes in a height balanced tree of height h is $2^{h+1}-1$ (upper bound)

What is the smallest number of nodes in a height balanced binary tree?

for height = 0 • $M_n = 1$

= 1 $M_n = 2$

= n

$M_n = M_{n-2} + M_{n-1} + 1$

$\log n$ time complexity

2-29-12

max nodes based on height $2^h \rightarrow 2^{h+1} - 1$

min nodes " " " $m_n = m_{n-1} + m_{n-2} + 1$

close to Fibinachi

$$M_n \approx \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n + 1$$

height of tree $\log_A m_n = \log_A \left(\frac{1}{\sqrt{5}} A^n + 1 \right)$ $A = \frac{1+\sqrt{5}}{2}$ global ratio

$$\approx \log_n \left(\frac{1}{\sqrt{5}} A^n \right)$$

$$\approx \log_a \frac{1}{\sqrt{5}} + \log_A A^n$$

$$\approx -1.673 + n$$

$$n \approx \frac{\log m_n}{\log A} = 1.44 (\log m_n)$$

$$n \in O(\log M_n)$$

Therefore, algorithms on height balanced binary trees that run proportional to the height n are in $O(n)$

book fixes

pg. 302 - bottom of page.

$n \approx 1.44 \log M_n$

M_n

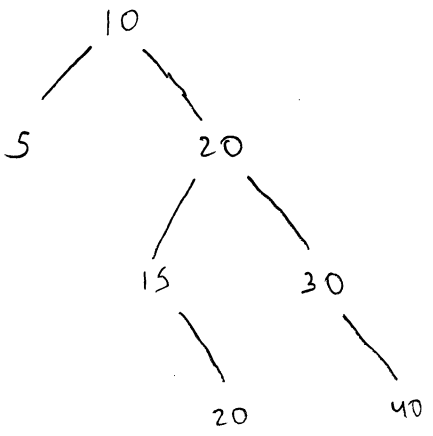
$\lceil \log n \rceil$ - minimum $\log M$

$O(\lceil \log n \rceil)$ $O(n)$

10, 20, 30, 5, 15, 40, 20

<= left

> right



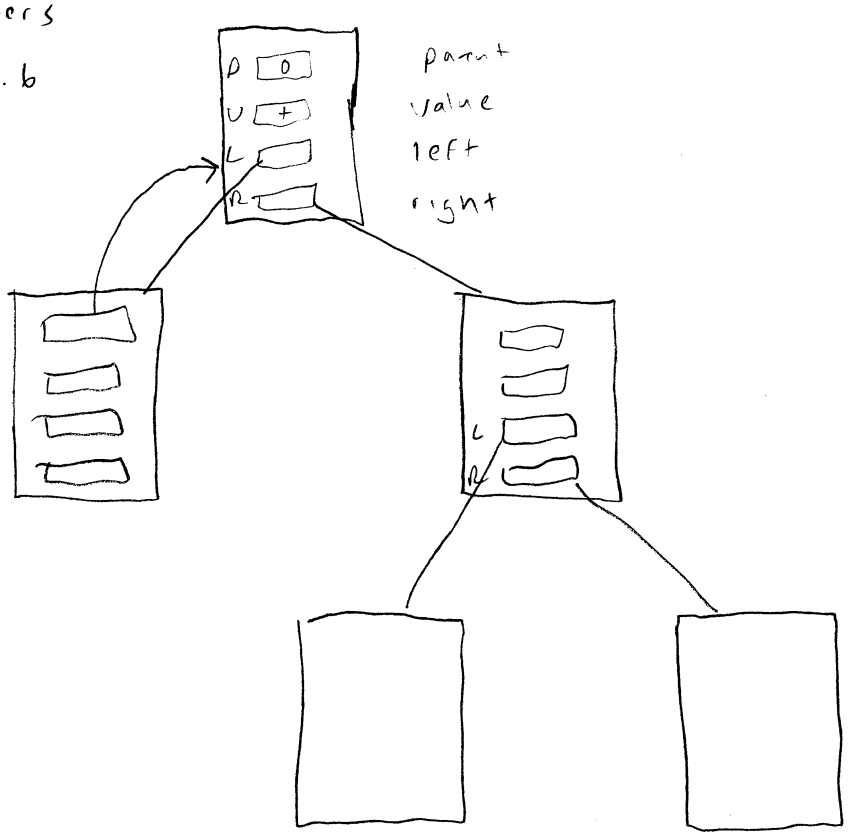
not height balanced.

Chapter 14 AVL Trees

Binary Tree Representation

① use pointers

fig 13.6



Node Class - template

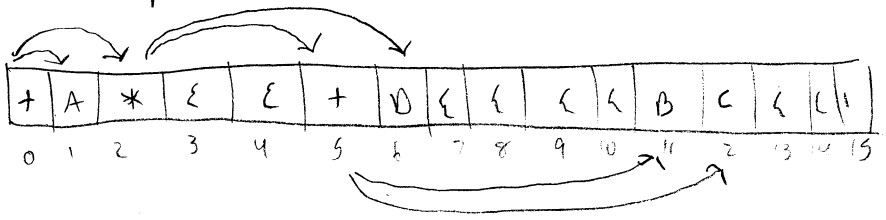
```
template < class T >
class Node {
  Public:
    T value
    Node<T> * parent
    Node <T> * left_child.
    Node<T> * right_child.
    Node (T x) { value = x; parent = leftchild = rightchild = 0; }
    copy constructor
    default constructor.
```

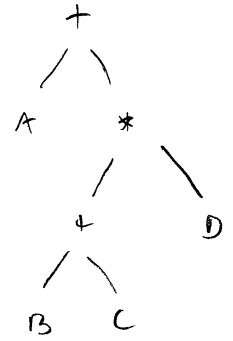
② Use a vector → vector <char> V

a. root is in V[0]

B. children of V[i] are in V[2i+1] & V[2i+2]

or parent of V[i] is at V[(i-1)/2]





{ = not used.

read chapter 12 & 13 & Deque

Sets & Multisets -

- must be unique
- no order

$A = \{30, 10, 40, 20\}$

$B = \{5, 40, 20, 15\}$

not ordered but

unique elements

$A \cup B = \{30, 10, 40, 20, 5, 15\}$

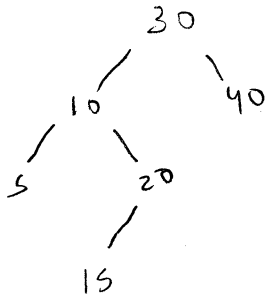
$A \cap B = \{40, 20\}$

In STL, both set and multiset are ordered.

Set has unique elements, multisets allows duplicates

↓

bag



LNR traversal, left node right

5, 10, 15, 20, 30, 40

pg. 109 operations on sets and their time complexities. O(logn)

```
#include <set>
set<int> s,t,v;
s.insert(30)
s.insert(10)
s.insert(40)
s.insert(20)
cout << s.count(10)  // 1 occurances
cout << s.erase(10)  // 1 erased.
  "  "  s.count(10)  // 0 occurances
s.insert(10)
t.insert(5)
   "   (40)
   "   (20)
   "   (15)
```

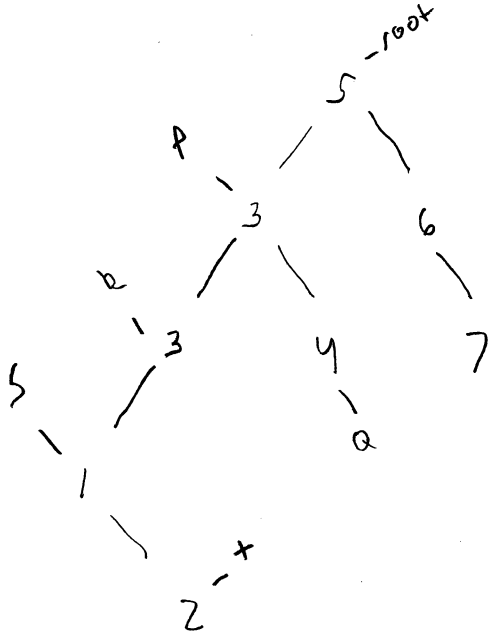
```
multiset<int> m
  m.insert(30)
  "  "  10
  "  "  40
  "  "  20
  "  "  10
  cout << m.count(10) // 2 occurances
cout << m.erase(10) // 2 erased.
  "  "  m.count(10) // 0 occurances.
```

Union of sets

```
insert_iterator< set<int> > inserter (v, v.begin)
                  ^ space.        ^ object
set_union (s.begin(), s.end(), t.begin(), t.end(), inserter)
set<int>::iterator i
for (i = v.begin ; i != v.end ; i++)
    cout << *i
```

3-5-12

Sets & Binary Trees continued....



CNR

1, 2, 3$_1$, 3$_2$, 4, 5, 6, 7

lower bound

root → lb(3, end) return result

p → lb(3, end) return result

r → lb(3, end) return setIterator(2)

5 → lb(3, end) end

+ → lb(3, end) end

upper bound

root → ub(3, end)

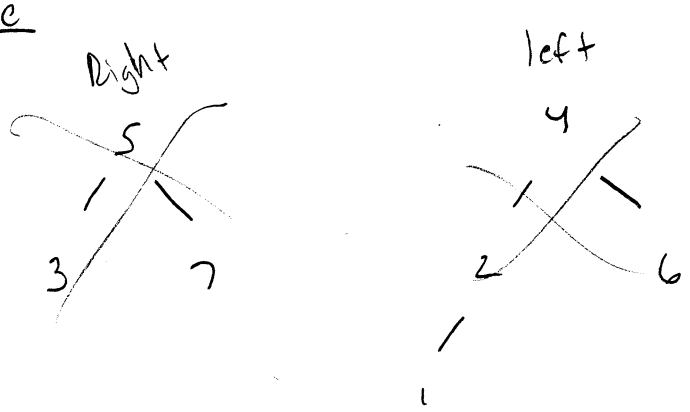
p → ub(3, end)

q → ub(3, end)

since no left child

return setIterator this

merge



right → leftchild, goes down left side of right tree

In order traversal LNR

Post
LRN

```
void inorder (node<T> *current)
{
    if (current != 0)
        inorder (current -> leftchild);
        cout << current -> value;
        inorder (current -> rightchild)
```

iterators

set<int>

set<int>::iterator i = s.begin() s.begin() perform left side.

left slide

3-7-12

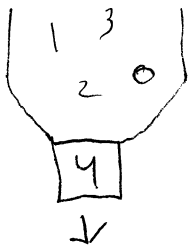
Priority Queues

need complete binary trees.

need # include <queue>

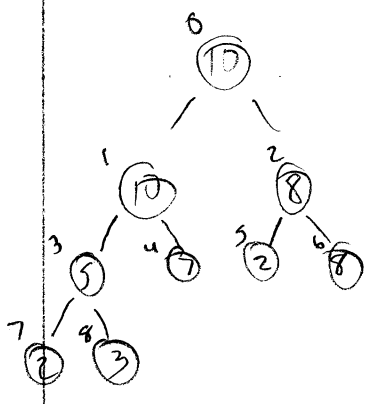
```
priority_queue<int> pq
pq.push(2)
pq.push(4)
   //  (3)
       (1).
cout << pq.top() // 4
pq.pop()
cout << pq.top() // 3
```

implementation: binary heap

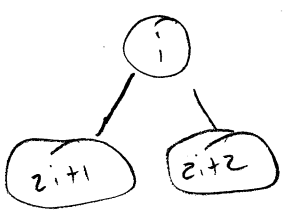


Complete binary trees in an array or vector

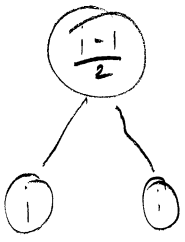
Every node greater than or equal to children.



10	10	8	5	7	2	8	2	3
0	1	2	3	4	5	6	7	8



for any i



for any i

newElement = 30

c.push_back(newElement)

if 30 > parent
swap

larger of two children on percelate down.

last (stop-start) - 1

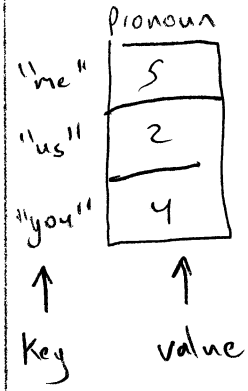
(heap size / 2) - 1
Starting position where might not be in heap property.

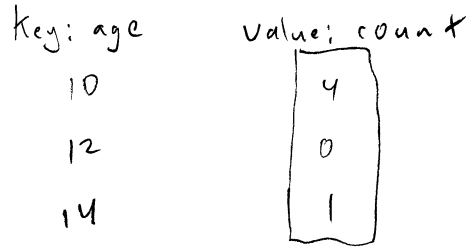
3-12-12

Maps & Multimaps

- maps are built on top of sets, chapter 16

Ordered collection of key-value pairs ← data structure.
An index data structure
Other names are associative memory or dictionary





key is ordered.

map - key must be unique.
multimap - key can be duplicates.

space.

```
Set < Pair<T1, T2> >
```

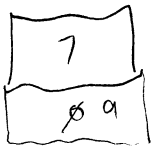
```
cout << pronoun["you"]
```

```
#include <map>

map <string, int> m
int x = m["Henry"];
```

→ Henry doesn't exist when called

→ x=0




```
m["Harry"] = 7
m["Henry"] = 9
```

- if value doesn't exist, [] will create it with default value.

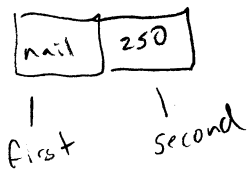
- map is not limited to binary search trees.

hardware

nail	100
saw	3
nail	250
hammer	2
saw	4
hammer	7
nail	1000

```
void readItems (map<string, int> & m)

    String word;
    int val;
    While (cin >> word >> val)
        m[word] += val;
```




```
int main()
map <string, int> tbl;
readItems (tbl);
```

→ int total=0

```
map <string, int>:: const_iterator p;
 for (p= table.begin(); p != table.end(); p++) {
        total += p→second
        cout << p→first << "\t" << p→second
}
```

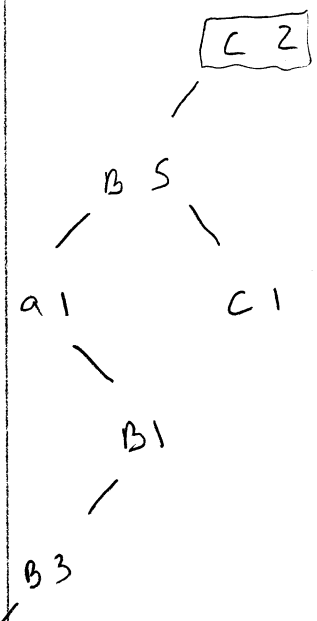
cout << "total " total

3-14-12

maps & multimaps continued

```
multimap<string, int> m;
m.insert( pair<string, int>("c", 2));
m.insert( pair<string, int>("b", 5));
```

a 1
b 1
b 3
c 1
b 1



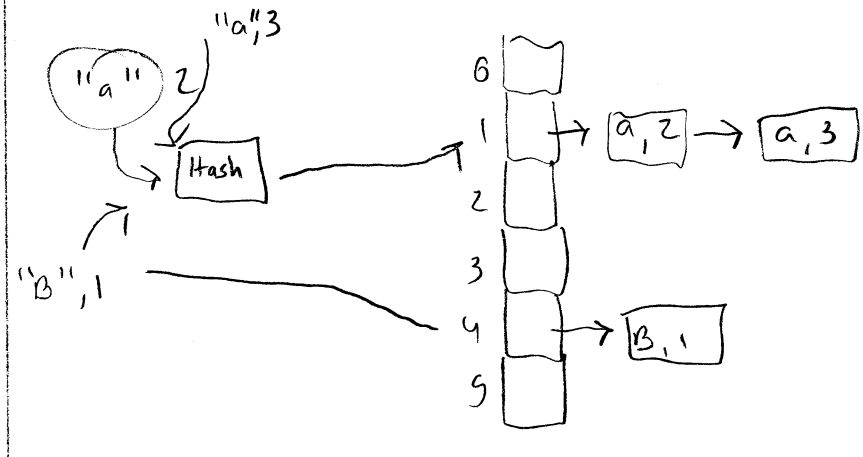

```
cout << m.count("c"); // 2
// cout << m["c"]; not defined with
                   multimap, is for map.
```

```
multimap<string, int>:: iterator it
for (it=m.begin(); it != m.end(); it++)
    cout << it->first << " " << it->second << endl;
```

a 1
b 1
b 3
b 1
b 5
c 1
c 2

```
it = m.find ("c")
cout << it->second;  1
```

Hash Table



map lab

find call find from set

erase call erase from set.

parent::erase()

String friends.

Chapter 17 Hash Tables

- use a hash function (O(1)) to
  store and look-up values in a hash table.

Ex. Hash table → vector<string> hash_table(10);

hash_function → int hfunc(const string &s)
return (s[2]-'a') %10;

Problems:

- what if name didn't have the 3rd character?
- what if there are two names that "map" or hash to the same place? (same 3rd letter)
- what if more then 10 names?
- what if our hash function is not perfect?
  This is always the case.

Solutions

use buckets, every location can hold more than one name
try to improve your hash function

Time complexity deteriorates to O(n)

```
class item {
public:
    Item (const string & v) : value(v), nex(0) { }
    String value;
    Item * next;
};

  vector < Item *> bucket (10);
```

3-19-12

Debugger — keeps symbols.

g++ -g testl.cpp

gdb a.out

find line number.

break testl.cpp:19

break Deque.h:77

run

p toprint ! p saved=size

Hash tables continued.

use prime number, number of buckets.

hash with list.

hash function returns unsigned int.