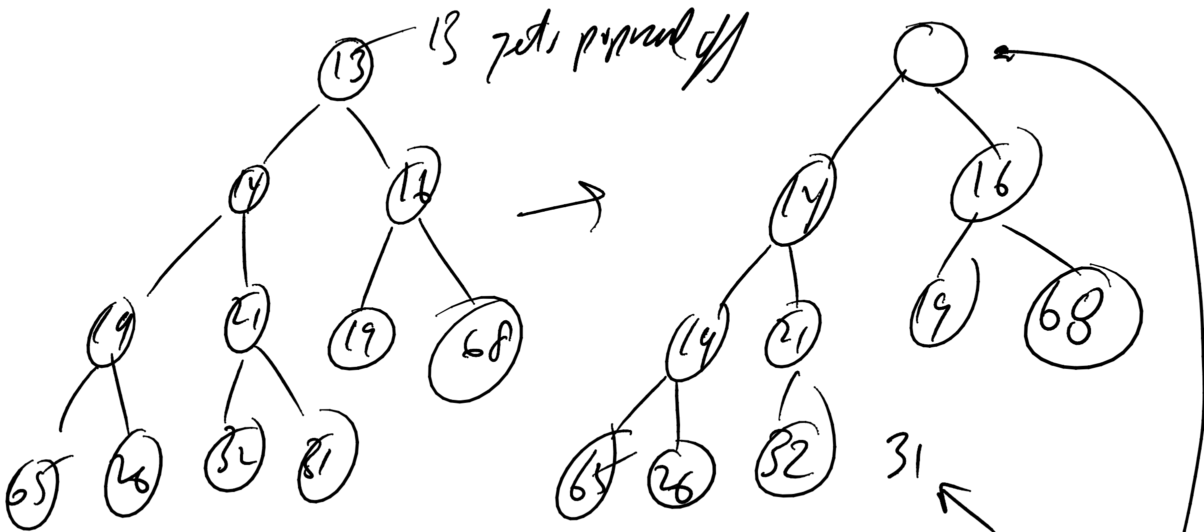
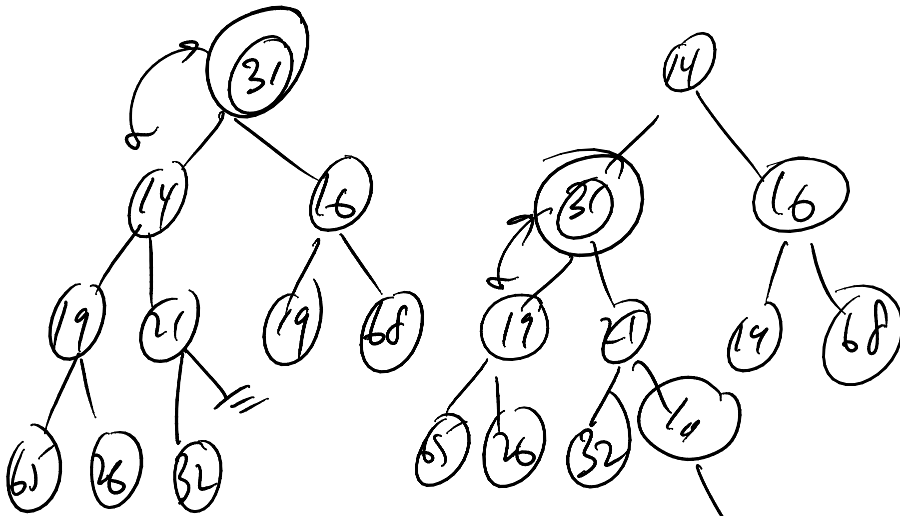
Thursday, September 17, 2009
3:33 PM

percolate down: (used after deleteMin)



sitting in temp or in

issue becomes where to insert 31?



...

the gotcha:
last node may or may not have the right child
→ requires an extra test

---

what is buildHeap?
used in heap constructor with an array of

- used in heap constructor with an array of elements as argument
- could not assume the array was in binheap format, i.e., just an array of numbers,

so buildheap: for each element of array
insert(element)

Thursday, September 17, 2009
3:46 PM

- with our List implementation, you should be able to use this for any kind of object,

  e.g.,   List<Pairs> plist;

where Pairs is :

```
class Pairs {
      Pair(float ix=0.0, char ic='a') : \
            x(ix), c(ic)  { };
      bool operator< (const Pairs& rhs)
            { return( c < rhs.c); }
      bool operator> (const Pairs& rhs)
            { return (c > rhs.c);}
      :
      private:
         float x
         char c;
}
```

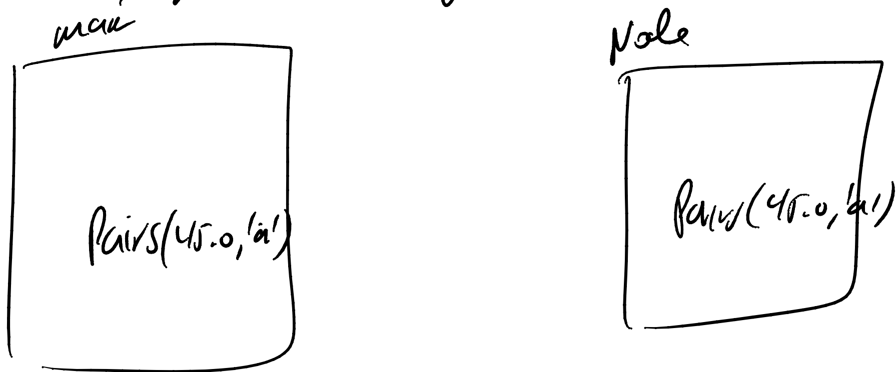
- using our List, this should work :

```
plist.push_back( Pairs( 45.0,'a'));
```

- important note on this :
  - we have a list of Pairs objects, each contains a float & a char
  - when you write plist.push_back(Pairs(45.0,'a')) what happens?
    - ... [illegible] ... const T& rhs) is called

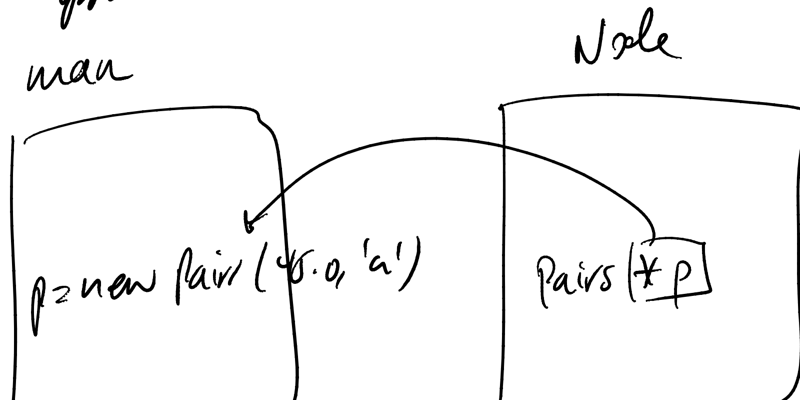
- eventually insert(iterator, const T & rhs) is called
- this calls Node constructor new Node (rhs, p->prev, p)
- Node (const T & d --) :\
      data(d);
  this calls Pairs copy constructor (DEEP COPY)

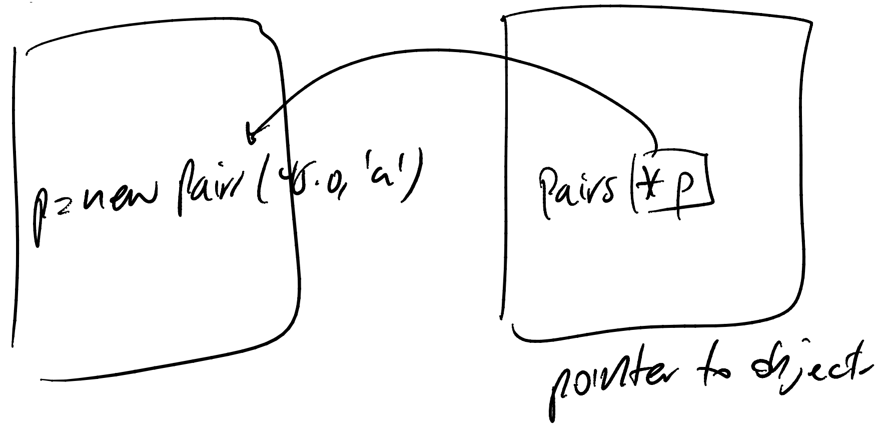
  we now have a copy of the original object
  (this is very important for memory performance)



- not a memory leak, just redundant ok
- Pairs in this case is a "light" object (5 bytes, big deal)
- what if it were a "heavy" object?
  lots of floats, ints, lots of stuff etc.

- a better approach:





- moral of the story: for large objects, use list of pointers, e.g.

```
list<Pairs *>   plist;
Pairs           *pp = NULL;
```

then

```
pp = new Pair(45.0, 'a');
list.push_back(pp);
```

2 things to remember

1. you used new to allocate memory, you must use delete to free it up when obj is taken off list (otherwise memory leak)
2. when iterating thru list, *itr is pointer to data

e.g. `(*itr)->getX()`

Thursday, September 17, 2009
4:03 PM

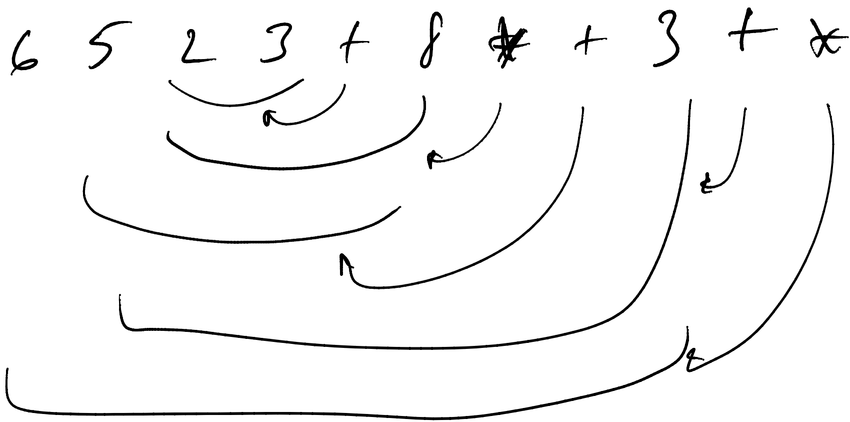
List applications: popular one is list acting like a stack

- basic ops: push & pop

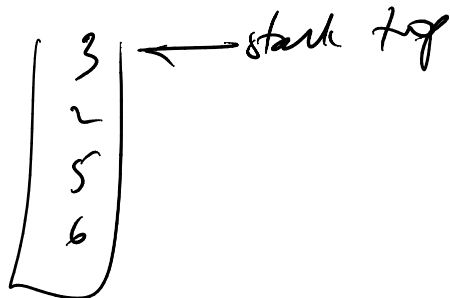
- your doubly-linked list directly provides these.
(push_back, pop_back, back())

- reverse-polish calculator:

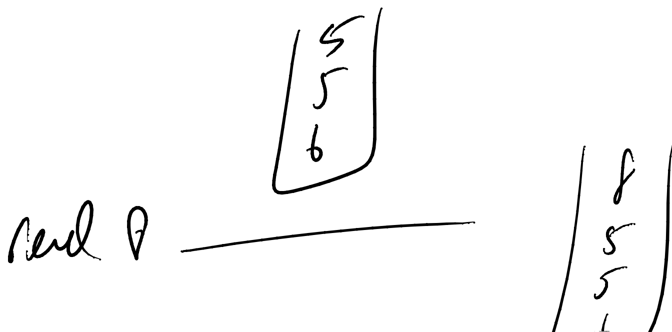
6 5 2 3 + 8 * + 3 + *



- if a number (operand), push on stack
- if operator, pop last 2 things off, do the op, push res



read +, pop off 3, 2 do 2 + 3 push result



read 8 — [6]

[8 / 5 / 5 / 6]

'*' : 5×8

[40 / 5 / 6]

'+' :

[45 / 6]

3 :

[3 / 45 / 6]

'+' :

[48 / 6]

'*' :

[288]

input is coming in from std::cin

when to pop results?
when you get '\n'

when to quit?
when you get CTRL-D

-1

Thursday, September 17, 2009
4:12 PM

- what about infix notation?
- use one stack to push operator, another to store postfix notation

$a + b * c + (d * e + f) * g$

→ $a b c * + d e * f + g * +$

Thursday, September 17, 2009
4:14 PM

How to parse input line:

87␣19␣+
↑

```
int num;
char c;
while (c = std::cin.get()) {
    if (isdigit(c)) {      // from #include <ctype.h>
        std::cin.putback(c);
        std::cin >> num;
    } else {
        switch(c) {
            case -1:   // ctrl-D (EOD)
                return 0;
            case '*':   // pop, mult, push
                break;
            :
            case '\n':   // pop stack, output to cout
        }
    }
```