main.cpp and class list_t (lab7)

Wednesday, October 06, 2010
8:58 AM

— don't forget to update Makefile

```
CC = g++

.C.o:

.cpp.o:
```

} can stay but we now need

"suffix rule"

---

list.h:

```
#ifndef LIST_H
#define LIST_H

class list_t
{
  ;
};

#endif
```

→ this allows you to #include "list.h" multiple times in a .cpp file — you shouldn't really need to but this forgives the sloppiness.

#endif → needs to be unique

- rule of thumb: for any C++ object, start with interface, i.e, the class definition (in header file) → how you want the class to be used

- then "fill in" the implementation (what's in the .cpp file)

simple functions, like one-liners such as accessors or mutators can be given in interface.

Wednesday, October 06, 2010
9:25 AM

class list_t

at bottom, we have
private:
link_t *first;
link_t *last;
link_t *current;

{

private:

struct link_t
{
    link_t *next;
    void   *data;

    link_t(void * idata = NULL) :
        next(NULL), data(idata) {};
};

setting aside up for data structures course (212)

incoming optional input: init argument

line continuation

public:
// constructors (overloaded)
list_t() { first = last = current = NULL; }

— or —

list_t() :
    first(NULL), last(NULL), current(NULL) {};

// copy constructor: don't really make sense here — would need DEEP copies

list_t(const list& rhs) :
    first(rhs.first),

list_t alist;
list_t blist;

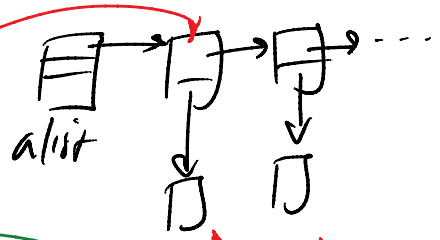
first(rhs.first), \
last(rhs.last), \
current(rhs.current) \
{};

this is the "head idea"

blist
points at rhs's data

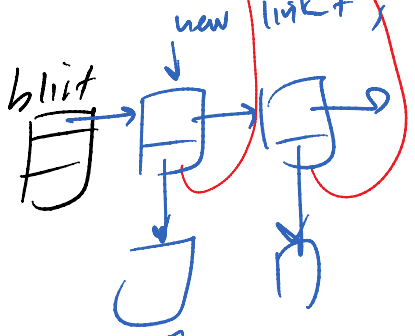
list_t alist;
list_t blist;

alist.add(...); ...



rhs

blist = alist



head idea
in real:
don't
copy
*data,
just
point to
existing object

deep copies

new *data objects
(copies of current
or alist)

```
// destructor
~list_t() { del(); }

// members
void add(void *);
void del();
void del_front_link();
void reset() { current = first; }
bool empty() { return first == NULL ? true : false; }
```

in list.cpp,
just need code
for

```
bool not_end { return current != NULL ? 1 : 0; }
int next_link { current = current -> next; }
void * get_data() { return (current -> data); }

private:
link_t *first;
"      *last;
"      *current;
};
```

---

list.cpp

```
#include <iostream>
#include "list.h"

void list_t::add (void *data)
{
  ~

  }

  ;

void list_t::del ()
```