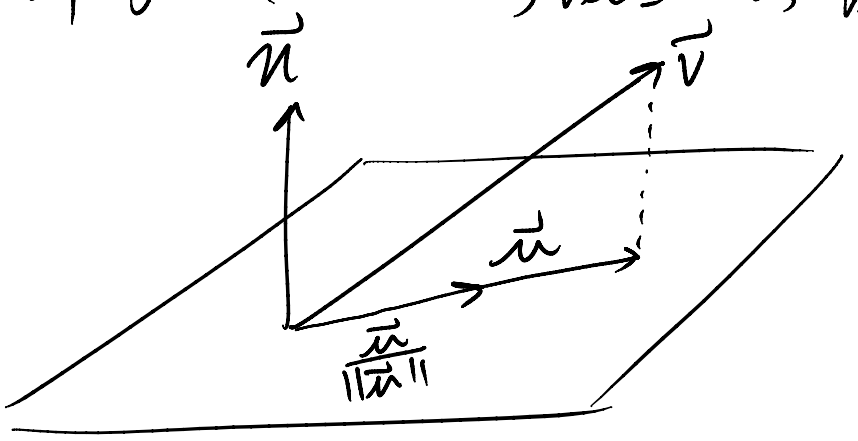
Building a generic linked list

Friday, August 27, 2010
10:25 AM

plane normal

```
void vec_project(vec_t n, vec_t v, vec_t u)
```



↑ result: the projection

"orthogonal projection u of vector v onto plane defined by n (normal) is $u = v - (v \cdot n)n$"

dot prod

scale

vector diff a-b (a + (-b))

```
{
    vec_t w; // work vector

    vec_scale(-vec_dot(v,n),n,w);   // -(v·n)n
                                    // scalar
    vec_sum(v,w,u);

    vec_unit(u,u); // normalize u
}
```

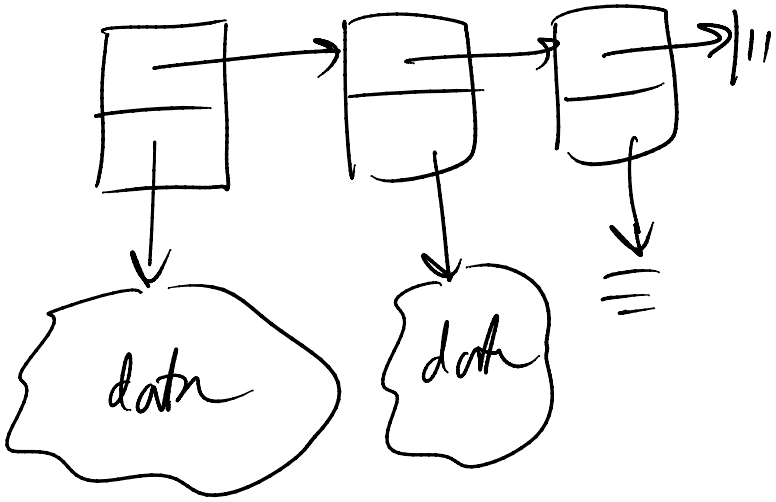
Monday, August 30, 2010
9:10 AM

- Building our linked list "manager" (list object, or class) — abstract data type (ADT) — in C++
- use it for list of objects, materials, lights.
- list is generic
- The list itself is composed of 2 data types: list header ("the list"), list link (node of the list)

```
typedef struct link_type
{
    struct link_type   *next;
    void               *data;
} link_t;
```

void* data;
void *data;

2 ptrs — void *data1, *data2
1 ptr, 1 void (?) — void* data1, data2



- very first function you should (always) write: object constructor (memory alloc & init)

- idea; usage:

```
link_t *links = link_init(data);
```

(void * / fill this in)

```
#include <assert.h>
#include "list.h"
link_t* link_init(void* data)
{
    link_t* link = (link_t*)malloc(sizeof(link_t));
    assert(link != NULL);
        -or-
    link_t * link = NULL;
```

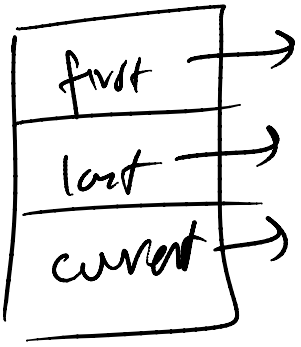
```
link_t * link = NULL;

if ( (link = (link_t*) malloc(sizeof (link_t))) == NULL){
    fprintf(stderr, "out of memory\n");
    exit(0);
}

link->next = NULL;
link->data = data;
return (link);
} // link constructor.
```

data greater than

Monday, August 30, 2010
9:28 AM

list header:

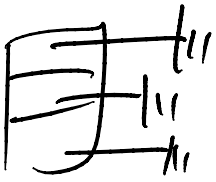
one of these per list



usage:

```
list_t *elist=NULL;
elist = list_init();
```

```
list_t * list_init(void)
{
    list_t  *hdr = NULL;
    hdr = malloc(sizeof(list_t));
    assert(hdr != NULL);
    // NULL ptrs indicate empty list
    hdr->first = hdr->last = hdr->current = NULL;
    return(hdr);
}
```



```
int list_empty(list_t *list)
{
    if(list->first == NULL)
```

return 1;

else

return 0;

OK

return list->first == NULL ? 1 : 0;

}

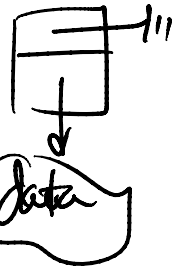
if ? | |

if else

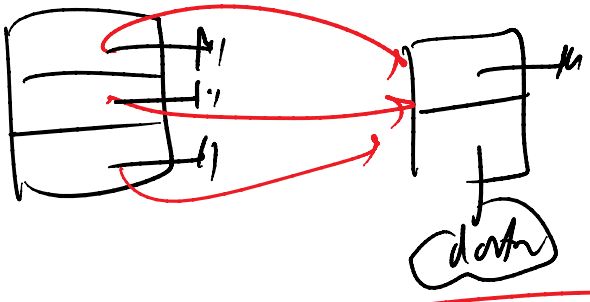
---

list_add (list_t *list, void *data)

1. get a link :



2. if list empty,



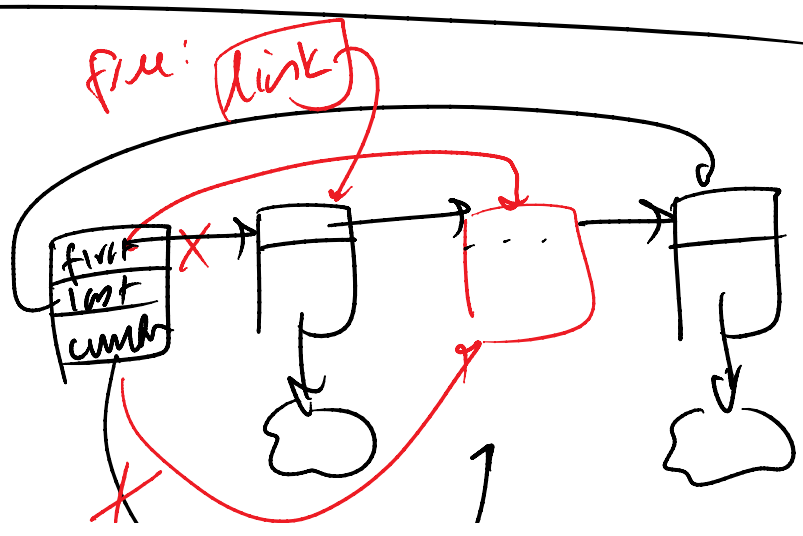
else


```
link_t *link = link_init(data);
if(list_empty(list)) {
    list->first = list->last = list->current
                = link;
} else {
    list->last->next = link;
    list->last = link;
}   // don't touch current - points to first.
```

---

deletion

```
our list;
```



delete from front of list

```
void list_fdel(list_t *list)
{
        link_t  *link;

    assert(list->first != NULL)

    link = list->first;
    list->first = list->first->next;
    list_reset(list); // sets current to front

    free(link->data) // freeing up user data

    free(link);
}


void list_del(list_t *list)
{
        list_reset(list); // list->cnext = list->first;
```

```
while (list_not_end(list))
    list_fdel(list);
}

list_not_end:
return list->current != NULL?
    1 : 0;
```