Lab 10: object_t & plane_t

- inheritance the easy way
- objective of lab 10 is to read plane.txt file just as before (lab 4 A Hack)
- use list_t *mats =

  [(list_t *)[list_t alloc] init];
- same for list_t * objs;

- same reading of 'token'
- use C-style I/O string

```
fscanf(stdin, "%s", token)
if (!strcmp(token, "material"))
{
    mat = [[material_t *
        [material_t alloc] init];
    [mat read: stdin];
    [mats add: mat];
}
```

- same for "plane":

```
obj = [(plane_t *)
    [plane_t alloc] init];
[obj settype:
    [[NSMutableString alloc]
    initWithUTF8String:
        token]];
[obj read:stdin :mats];
[objs add:obj];
```

NSString that can change

object_t *

- note that

```
object_t *obj = nil;
```

obj is declared as object_t

- but then allocated as plane_t

```
obj = [[(plane_t *) plane_t alloc]
        init];
```

- to output model:

```
[mats write: stdout];
[objs write: stdout];
```

- so far, usage looks pretty good — one chief advantage of OOP
- just need to know relationship between object and plane

- object_t is the parent class, the virtual

  object — ray cannot intersect object_t

- however, all derived objects must implement hits() function ≡ must implement virtual function

→ conform to @protocol

- object.h:

```
@protocol intersecting
-(double) hits:
    (vec_t *) _pos:
    (vec_t *) _dir:
    (vec_t **) _hit:
    (vec_t **) _N;
@end
```

notice pass-by-reference pointers - to - pointers

- still in object.h :

```
@interface object : NSObject
{
    int cookie;
    NSString *type;
    NSString *name;
    material_t *mat;
    vec_t *last_hit;
    vec_t *last_N;
};
```

probably won't be used

- member functions:

setType, setName } setters (mutators)

getName, getType } getters (accessors)

matches } query

read, write } I/O

@end

- plane.h:

```
@interface plane_t : object_t <intersecting>
{
    vec_t *normal;
    vec_t *point;
    double ndotq;
}
```

protocol

parent (inherit from)

```
-(id) initi

-(void) read: (FILE *) _in:
                (list_t *) mats;

-(void) write: (FILE *) _f;

@end
```

```
- plane init:
    if(!(self = [super init]))
        return nil;
```

calls object_t's init function

```
    normal = [[rect alloc] init];
    point = [[rect alloc] init];
    ndotf = 0.0;
    return self;
```

- object_t init:

```
if(!(self=[super init]))
    return nil;
cookie = OBJ_COOKIE;
name = [[NSString alloc]init];
type = [[NSString alloc] init];
mat = nil; // just a pointer
           no mem alloc
last_hit = [[vec_t alloc] init];
last_N = [[vec_t alloc] init];
return self;
```

— Remember that back in main() we had:

```
if(!strcmp(token,"plane"))
{ obj=[(plane_t*)[plane_t alloc]
        init];
  str=[[NSSMutableString
    alloc]initWithUTF8String:
      token];
  [obj settype:str];
  [obj read:stdin:mnats];
  [objs add:obj];
}
```

ADD TO LIST

```
-(void) read: (FILE *)_in:
              (list_t *) mats
{
    char   c, _name[256];
    int    count;
    NSMutableString *str;
[super read: _in: mats];
```

get parent (object_t) to read itself in & find mat pointer

then loop until we get '}':

```
while((c=fgetc(_in))!=EOF
&& c!='}') {
    fputc(c,_in);
    count=fscanf(_in,"%s",_name);
    if(!strcmp(_name,"normal"))
        [normal read:_in];
    if(!strcmp(_name,"point"))
        [point read:_in];
    while((c=fgetc(_in)!=EOF
    && c!='\n');
}
```

- after reading in file, normalize plane normal & set last_N, calc ndotf:

```
[normal autorelease];
```
} avoid mem. leak

```
normal = [[normal unit] retain];
```

prevent seg fault due to autorelease called by unit

```
[last_N autorelease];
last_N = [normal copy];
ndotf = [point dot: normal];
```

- What about object_t's load:

```
-(void) read:(FILE *)_in:
            (list_t *) mats
{
    char    c, _name[256];
    int     count;
    NSMutableString *str;
    count = fscanf(_in, "%s", _name);
    [self setName:[[NSString alloc]
        initWithUTF8String:_name]];
```

Then read in material

```
while((c=fgetc(_in))!=EOF
&& c!='{') {
```

= read until first {

```
count=fscanf(_in,"%s",_name);
if(!strcmp(_name,"material"))
{
  count=fscanf(_in,"%s",_name);
  str = [[NSMutableString alloc]
    initWithUTF8String:_name]];
  mat = [mats getbyname:str];
}
while((c=fgetc(_in)!=EOF
&& c!='\n');
```

- similarly for write:
- planet's write method first calls parent's write:

```
[super write:_out];
```

then writes out normal, point

```
[normal write:_out:"normal"];
[point write:_out:"point"];
fprintf(_out,"}\n\n");
```

- note that obj already added to objs list (done in main)

- eventually done in model.m

- the main thing to note here is that memory representation is hidden, but still the same internally i

object_t

int cookie
NSString *type
NSString *name
material_t *mat
vec_t *last_hit
vec_t *last_N

plane_t

vec_t *normal
vec_t *point
double dot

MIDTERM

- 10 questions
- weighted unequally
- C (no Obj-C)
- OOP concepts, e.g.,
  - inheritance
  - polymorphism
  - dynamic binding
  - encapsulation
- linear algebra (e.g. vector math)

- C:
  - "what does this program do?"
  - "write a function to ..."
  - pointers
  - linked lists
  - pointers to functions