From C++ to Objective-C

(if you want to code for the iphone, ipad...)

- good writeup by Pierre Chatelier

- main differences:
  - every object is dynamically allocated
  - can't say:

    object_t object;

    must be:

    object_t* object;

- can init to nil, a special NULL object pointer or call object's constructor

```
object_t *object =
    [(object_t *)[object_t alloc] init];
```

cast <> compiler knows which init to call

Message to class

init member fxn

[object doSomething] ≡

object → doSomething();

- arguments passed in via colon-delimited list:

[object do: arg1 : arg2]

(can add arg labels)

≡ object → do(arg1, arg2);

- object class definition in .h file.

```
@interface vec_t : NSObject
{
    double vec[3];
}

// constructors
-(id) init: (double) x;
-(id) init: (double)x : (double) y;
-(id) init: (double)x : (double) y : (double) z;
```

base class

data members listed first

```
//accessor/mutator
-(double) get:(int)i j
-(void) set:(int)i:(double) d;

// "operators"
-(vec_t *) plus:(vec_t *)rhs;

@end
```

not passing by reference, pointer only

- implementation in .m file

```
#import <vector.h>          ← handles

@implementation Vec_t       #ifndef VEC_T
                            #define VEC_T
-(id) init : (double)x      #endif
{ return [self init : x :: 0.0 : 0.0]; }

-(id) init : (double)x : ... (double)z
{ if ( ! (self = [super init]) )
      return nil;
  vec[0] = x; vec[1] = y; vec[2] = z;
  return self;
}
```

class member fcts, + this

```
-(double) get: (int) i
    { return vec[i]; }
```

- example usage of vec_t class

```
int main(int argc, const char *
                        argv[])
{
    NSAutoreleasePool *pool =
        [[NSAutoreleasePool alloc]
                        init];
    vec_t *v1 =
        [[vec_t alloc] init];
    vec_t *v2 = [[vec_t alloc] init];
    vec_t *v3 = nil;
```

v3 = [v2 minus: v1];

- how to implement this ?
  minus should return allocated memory
- (vec_t *) minus: (vec_t *) rhs

{
    vec_t *result = [[vec_t alloc] init];
    int i;
    for(i=0; i<3; i++)
[result set: i : v1[i] - [rhs get: i]];
    return result;
} returning without ret.

- with

```
v3 = [v2 minus: v1];
```

need to remember to release (free) memory

```
[v3 release];
```

- what about "transient usage" when writing:

```
[[v2 minus: v1] writeTo: stderr]
```

what happens here? MEM LEAK!

So the right thing to do is:

return [result autorelease];

↑ this says release memory later (after usage)

- but then what about

v3 = [v2 minus: v1];

later in the code v3 will be released unless you say:

v3 = [[v2 minus: v1] retain];

- at the end of main:

```
[u1 release];
[u3 release];
[pool drain];
}
```

free up auto-released memory

- can have several such pools to limit mem usage

- Mem. mgmt. is important in obj-c.

- inheritance : ← virtual void func.

@protocol interesting

- (double) hits : (Ray_t *)

@end pos : (Ray_t *) div;

@interface object_t : NSObject

{

  init cookie;

  NSString *type; // e.g. "plane"

  NSString *name; // e.g. "floor"

  material_t *mat;

}

- (id) init;

- (void) read.. (FILE *) in ...

@end

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

plane_t must implement hits function

- parallelization of the code:
- can use omp again:
  each thread needs
  its own autorelease pool;
  (doesn't seem to work
  in GNUstep; seems
  ok on Mac's with
  objc-(2.0)