What's a pointer, why good, why bad?

- Pointer is a memory address, it's an indirect reference to memory or an object.
  - Difference between Foo and Foo *
    - One is a pointee (object) and one is a pointer (reference)
    - Pointers to objects helpful: inheritance, linked structures
- Pointers force us to think about the machine and memory
  - Knowledge is powerful, but freedom from it liberating
- Pointers allow us to work at a lower level, but permit inheritance and a higher level of design/programming
  - Built-in array and tvector, C-style string and <string>

Pointers, Memory, Abstractions

- A pointer is a variable/value that is a memory address
  - Addresses like 1, 2, 3, ..., 0x0024ab03
    - Hexadecimal or base-16 digit represents 4 bits
    - Character is 8 bits, integer is 32 bits, double 64 bits (ymmv)
  - Every object is stored somewhere in memory, typically we can ignore where
    - double x = 32.6;
    - int y = 18;
    - string s = "hello";
  - The string variable s may be "same size" as double x
    - Storage for the letters is elsewhere, string references it, so memory used by string is more, though size of s isn't

Pointers, Heap, Copies

- Memory allocated statically (auto) vs. on the dynamically (heap)
  - Static = auto = stack
  - Dynamic = heap
- Objects are copied in C++
  - Semantics: copy, don't share
- Pointers are copied (object not)
  - Semantics: object not copied, object is shared

Pointers basics and terminology

- new, dereference, selector operator, copy semantics
  - CD c1("Beatles", "Rubber Soul", 1965);
  - CD c2("Nirvana", "Nevermind", 1991);
  - CD * c3 = new CD("REM", "Reveal", 2001);
  - CD * c4; // what is the value of c4?
  - CD c5; // what is the value of c5?
  - cout << c1.title() << endl;
  - cout << c3->title() << endl;
  - cout << (*c3).title() << endl;
  - cout << c4->title() << endl;
  - cout << c5.title() << endl;
  - What happens if we print c4->title() before = c3 line?
**Pointer Rules of Advice**

- Always initialize pointer variables
  - Assign 0 or NULL
  - Call new and assign returned value
  - Initialize to value of an initialized pointer
- Never use the address-of operator &

```cpp
CD cl("Beatles", "Rubber Soul", 1965);
CD * c2 = &cl; // Don't do this!!!
CD * c3 = new CD("REM", "Reveal", 2001);
```

- Don't call new unless you want another object allocated

**Tracing new and next**

```cpp
struct Node
{
  string info;
  Node * next;
  Node(const string& s, Node * link)
  : info(s), next(link)
  
};
Node * p = new Node("apple", 0);
Node * q = new Node("cherry", p);
Node * r = new Node("lemon", 0);
q->next->next = r;
```

**Pointers, what not to do**

```cpp
Node * p = 0;
p->next = new Node("apple", 0);
Node * q;
q->next = new Node("berry", 0);
```

- What happens when a bad pointer is dereferenced?
  - NULL/0 pointers generate segmentation fault/error
  - Bogus values might cause error, might access memory
  - Which is better (Journal of Irreproducible results)?

- [http://cslibrary.stanford.edu/104/](http://cslibrary.stanford.edu/104/)

**John von Neumann**

“Anyone who attempts to generate random numbers by deterministic means is, of course, living in a state of sin.”

“There’s no sense in being precise when you don’t even know what you’re talking about.”

“There are two kinds of people in the world: Johnny von Neumann and the rest of us.”

Eugene Wigner, Noble Physicist