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
- Every variable is stored somewhere in memory, typically we can ignore where

```
double x = 32.6;
int y = 18;
string s = "hello";
```
- The string variable s is actually the same size as int y
  - Storage for the letters is elsewhere, string references it

- What about a permanent, but forwardable email address?

What’s a pointer, why good, why bad?

- Pointer is a memory address, it’s an indirect reference to memory or an object.
  - Rather than say we have an int, we say we have a pointer to an int
  - If x is an int, xptr can be a pointer to x
    - Same thing works with Date, Dice, Student, ...
    - Not much use to have pointer to int, but pointer to class!!
- Pointers force us to think about the machine and memory
  - Knowledge is powerful, 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>

Pointer 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;
CD c5;
```
- What happens if we print `c4->title()` on first line? Why?

What’s the point?

- What’s the difference between a vector of Dates and a vector of pointers to Dates? What about Courses, Students, etc.?
  - `tvector<Date> tv(1000);`
  - `tvector<Date *> tvp(1000);`
  - Which takes up more space? What are values in vectors?
  - What happens when we write
    - `tv[0] = tv[2];` // if we change tv[2], affect tv[0]?
    - `tvp[0] = tvp[3];` // change *(tvp[3]), affect tvp[0], *tvp[0]?
- Consider example of sorting by both name and age
  - Should we have two vectors of students?
  - Should we have two vectors of student pointers?
  - Is there a reason to prefer one to the other?
Thinking about pointersort.cpp

- The class Group uses a `tvector<Student>` myList
  - What changes if this is `tvector<Student *>` myList?
    - Changes to Group::add
    - Changes to Group::print
  - Other changes needed?
- What if we want to sort by age to print, leaving original order the same (why would we want to do this?)
  - Use another vector, sort it differently
  - Why is another vector a good idea?
  - Could use vector of indexes
    - Both are indirect references

The trouble with pointers

- Don’t use the address-of operator, &

```cpp
define Dice * makeDie(int sides)
{
    return new Dice(sides);
}
```

What about the code below with different versions?

```cpp
Dice * cube = makeDie(4);
cout << cube->NumSides() << endl;
```

- Pointer Advice
  - Always initialize pointer variables, 0/NULL or new
    - 0/NULL means errors are reproducible
    - Possible to assign another pointer value too
  - Never use the address-of operator
  - Don’t call new unless you want another object allocated

Who is Alan Perlis?

- It is easier to write an incorrect program than to understand a correct one
- Simplicity does not preceed complexity, but follows it
- If you have a procedure with ten parameters you probably missed some
- If a listener nods his head when you’re explaining your program, wake him up
- Programming is an unnatural art
- Won first Turing award