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 except in C to understand arrays, but pointers to objects are very useful

- Pointers may 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 the 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 (yymm)
  - Every variable is stored somewhere in memory, typically we can ignore where

Pointers, Heap, Copies

- Memory allocated statically (auto) vs. on the dynamically (heap)
  - Static = auto = stack
  - Dynamic = heap

  Date ides(3,15,2002);
  Date * foolptr = new Date(4,1,2002);
  Date * x = foolptr;
  Date y = ides;

- 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?
  cout << c1.title() << endl;
  cout << c3->title() << endl;
  cout << (*c3).title() << endl;
  c5 = c2; c2.changeTitle("Incesticide");
  cout << c5.title() << endl;
  cout << c4->title() << endl;

- What happens if we print c4->title() on first line? Why?
What's the point? (e.g., sharing)

- What's the difference between a vector of Dates and a vector of pointers to Dates? What about Courses, Students, etc.?

```cpp
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?

The trouble with pointers

- Don’t use the address-of operator, &
  ```cpp
  Dice * makeDie(int sides) {
    Dice d(sides);
    return &d;
  }
  ```

- 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

Constructors/Destructors

- Every object created must be constructed
  - If no constructor is provided, one will be provided for you
  - If you have a non-default constructor, the default-default constructor is not automatically provided
- When subclass object constructed, all parent and up are too
  - Parent objects can be implicitly constructed via default constructor
  - Alternatively, explicit constructor must be called and it must be called in an initializer list
- Constructors initialize state and allocate resources
  - Resources can be dynamic objects, files, sockets, ...
  - Who (or what) de-allocates resources?

Destructors and Delete

- Objects are (or should be at most times) destructed when they’re no longer accessible or used
  - For static/automatic variables this happens when object goes out of scope
  - For heap-allocated variables this happens when the delete operator (analog of new) is called on a pointer to an object
    ```cpp
    Student * s = new Student("Joe");
    delete s; // return storage to heap
    ```
  - When object is destructed, the destructor function is called automatically: `Foo::~Foo() (...)`
- It’s easy to mess up when deleting, can’t delete the same object twice, can’t delete an object not allocated by new, ...
  - Yahoo story on never calling delete: too many problems!
Who is Alan Perlis?

- It is easier to write an incorrect program than to understand a correct one
- Simplicity does not precede 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 act
- Won first Turing award

http://www.cs.yale.edu/homes/perlis-alan/quotes.html

Problems with inheritance

- Consider the student example and burrito eating
  - CosmicStudent is a subclass of DukeStudent
    - What behavior changes in the new subclass?
  - What about a UNCStudent eating cosmic cantina food?
    - Can we have CosmicDukeStudent and CosmicUNCStudent?
    - Problems with this approach?
- Alternative to inheritance: use delegation (aka layering, composition)
  - Just like myEnergy is a state variable with different values, make myEater a state variable with different values
  - Delegate behavior to another object rather than implementing it directly

Delegation with school/student

- If there’s a class Eater, then what instance variable/field will a Student store to which eating behavior delegated?

```cpp
void Student::eat()
{
  myEater->doEat();
}
```

- How is the eater instance variable initialized?
- Could we adopt this approach for studying too?
- When is this approach better/worse?

Review/Preview: Anagrams/Jumbles

- Brute-force approach to finding anagrams/solving Jumbles
  - Brute-force often thought of as “lack of thought”
  - What if the better way requires too much thought?
  - What if there’s nothing better?

```
nelir, nelri, neir, nei, nler, lner, lnei, nlei, nrel, nler, nile, nre, reni, renl, leni, lenr, lni, rnen, nrel, nerl, ... lenir, lenri, leir, lein, ler, lerin, lerin
```

- What’s the problem here?
- Is there a better method?
Brute force? `permana.cpp`
```
// find anagram of word in wordSource
// list is a vector [0, 1, 2, ..., n]
Permuter p(list);
int count = 0;
string copy(word);// makes copy the right length
for(p.Init(); p.HasMore(); p.Next()) {
    p.Current(list);
    for(k=0; k < list.size(); k++) {
        copy[k] = word[list[k]];
    }
    if (wordSource.contains(copy)) {
        cout << "anagram of " << copy << endl;
        break;  // find first anagram only
    }
}
```

Quantifying brute force for anagrams
- On one machine make/test a word takes $10^{-5}$ seconds/word
  - $9!$ is 362,880: how long does this take?
  - What about a ten-letter word?
- We're willing to do some pre-processing to make the time to find anagrams quicker
  - Often find that some initialization/up-front time or cost saves in the long run
  - We need a better method than trying all possible permutations
  - What properties do words share that are anagrams?

Toward a faster anagram finder
- Words that are anagrams have the same letters; use a letter fingerprint or signature/histogram to help find anagrams
  - Count how many times each letter occurs:
    - "teacher" 1 0 1 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
    - "cheater" 1 0 1 0 2 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0
- Store words, but use fingerprint for comparison when searching for an anagram
  - How to compare fingerprints using operator `==`
  - How to compare fingerprints using operator `<`
- How do we make client programmers unaware of fingerprints? Should we do this?

Another anagram method
- Instead of fingerprint/histogram idea, use sorted form of word
  - "gable" and "bagel" both yield "abegl"
  - Anagrams share same sorted form
- Similarities/differences to histogram/fingerprint idea?
  - Both use canonical or normal/normalized form
  - Normalized form used for comparison, not for printing
  - When should this normal form be created?
- When is one method preferred over the other?
  - Big words, little words? Different alphabets? DNA vs English?
OO and C++ features we’ll use

- We’ll use an adapter or wrapper class called Anaword instead of a string
  - Clients can treat Anaword objects like strings, but the objects are better suited for finding anagrams than strings
  - The Anaword for “bear” prints as “bear” but compares to other Anaword objects as 11001000000000000100000000
- C++ allows us to overload operators to help, not necessary but good cosmetically
  - Relational operators == and <
    - What about other operators: >, <=, >=, and !=
  - Stream operator <<
- How should we implement overloaded operators?

Overloaded operators

- In C++ we can define what operator == and operator < mean for an object (and many other operators as well)
  - This is syntactically convenient when writing code
  - C++ details can be cumbersome (see Tapestry Howto E)
- In readwords4.cpp there are three overloaded operators
  - What about > and >= ?
  - What about printing, can we overload operator <= ?
  - Access to data for a Wcount object, simple because public, but what about a class?
- Overloaded operators are not necessary, syntactic sugar.

Overloaded operators (continued)

- Typically operators need access to internal state of an object
  - Relational operators for Date, string, BigInt?
  - Where is “internal state”?
- For technical reasons sometimes operators should not be member functions:
  - BigInt b = enterBigValue();
    - if (b < 2) …
    - if (2 > b) …
  - We’d like to use both if statements, only the first can be implemented using BigInt::operator < (why?)
- Use helper member functions: equals, less, toString
- Implement overloaded operators using helpers

Anaword objects with options

- Can we use different canonical forms in different contexts?
  - Could have Anaword, FingerPrintAnaword, SortAnaword
  - What possible issues arise? What behavior is different in subclasses?
    - If there’s no difference in behavior, don’t have subclasses
- Alternative, make canonical/normalize method a class
  - Turn a function/idea into a class, then let the class vary to encapsulate different methods
  - Normalization done at construction time or later
  - Where is normalizer object created? When?