Tools for programming

- **Syntactic details of control and variables**
  - If, while, switch, function calls
  - Scope and lifetime of variables

- **Arrays, strings, other containers (e.g., StringSet)**
  - Hold homogenous collection, support certain operations
  - String, e.g., is collection of char values

- **Object oriented and object based programming**
  - Encapsulation of state and behavior in classes
  - Data and the operations on it belong together
  - Object orientation requires inheritance/polymorphism
Toward understanding inheritance

- Consider original/old pixmap code for interacting with user
  - Several kinds of operations: load, store, invert, reflect, …
  - Each operation requires code spread over pixchoices.cpp
    - Edit working program, add code to code that already works
    - Not haphazard, but dangerous: if it ain’t broke, don’t touch
    - Initialization, selection of operation, implementation, …

- The open-closed principle of program building/design
  - Programs/classes should be open for extension, but closed to modification (Bertrand Meyer and Robert Martin)
  - How do we change code without altering it?
    - This is a goal, in practice we modify some, but minimize

- Inheritance lets us realize the principle
Use interfaces, not implementations

• See PixOperator in pixoperator.h, used in pixchoices.cpp
  ➢ Each subclass must implement operate method
    • Pure virtual in parent class, requires subclass implementation
    • Which virtual function determined at runtime

• We have parent, base, or super-class for interface
  ➢ Child or Subclasses extend the parent class (implement interface)
  ➢ Pointers to parent class, which can point to child object
    • I’m pointing at an animal (could be marsupial or bird)

• Child/sub classes don’t have access to private data/functions, but the data are there (alternative: use protected)
Syntactic details

- Functions in parent class that are changeable in subclasses must be declared as *virtual*
  - The “real” function is determined at runtime
  - In parent/base/super class make all functions virtual
  - We must have a virtual destructor that’s implemented

- For interface only functions, that have no default version
  - Pure virtual functions, aka abstract functions
  - Syntax requires = 0 in .h file
  - Can’t instantiate an object of an abstract (base) class
    - Doesn’t matter, really, we use pointers to objects
Conceptual Details

- **Some behavior is implemented in parent class**
  - Can be changed or overridden in subclasses, but we won’t do this in pixmap program

- **Some behavior specified in parent class, implemented in subs**
  - Pure virtual functions in parent class make it abstract

- **In C++ inheritance requires pointers or references**
  - No copies of objects passed around
  - Pointers required for storing in vectors, maps, etc.

- **Runtime determination of behavior, runtime allocation of storage**
What inheritance gets us (old/new)

```cpp
static string Operations [] = {
    "(load) an image ",
    "(save) an image ", ...
};

void PixChoices::operate(string op) {
    string s = LowerString(op);
    if (s == "load") {
        doLoad();
    } else if (s == "save") {
        doSave();
    } ...
}
void PixChoices::doLoad() {
    // code here
}
void PixChoices::doSave() {...}
```
Object based v. Object oriented

- **Original pixmap code in pixchoices.cpp**
  - All code in one file, advantages and disadvantages
  - Every pixmap operation in at least three places
  - What happens in “big” programs? What about testing?

- **New version has all code for operations in one place: subclass**
  - Easy to develop new operations: where does code go?
  - Small modification to pixchoices.cpp required, where?

- **Hard to see relationships**
  - Original could read one file
  - New version many files, maps, …
Pointers, Memory, Abstractions

- A pointer is the 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

```java
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;  // 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;
    c5 = c2;  c2.changeTitle("Incesticide");
    cout << c5.title() << endl;
    c4 = c3;  c3->changeTitle("Out of Time");
    cout << c4->title() << endl;

- 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.?

```cpp
tvector<Date> tv(1000);
tvector<Date *> tvp(1000);
```

- Which takes up more space? What are values in vectors?
- What happens when we write
  ```cpp
tv[0] = tv[2];  // if we change tv[2], affect tv[0]?
tvp[3] = new Date(12,19,2003);
tvp[0] = tvp[3];  // change *(tvp[3]), affect tvp[0], *tvp[0]?
  ```

- Compile time objects compared to runtime objects
  - For pointers, objects created at runtime, not compile time!
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