Admin

◊ Assign 2 handout missing page!
  ▪ PDF on web is complete or take a page to amend your paper copy
◊ Today's topics
  ▪ CS106 class library: Functions as data, client callbacks, recursion intro
◊ Reading
  ▪ Reader ch. 4 (today), ch. 5 (next)
◊ Got strep?
  ▪ No cafe handout today, but I will be in my office after class for a while

Specific plot functions

```c
const double Incr = .1;

void PlotSin(double start, double stop)
{
  double centerY = GetWindowHeight()/2.0;
  MovePen(start, centerY + sin(start));
  for (double x = start; x <= stop; x += Incr)
    LineTo(x, centerY + sin(x));
}

void PlotSqrt(double start, double stop)
{
  double centerY = GetWindowHeight()/2.0;
  MovePen(start, centerY + sqrt(start));
  for (double x = start; x <= stop; x += Incr)
    LineTo(x, centerY + sqrt(x));
}
```

◊ Code is identical, except for function invoked
  ▪ Let's unify!

Generic plot function

```c
void Plot(double start, double stop, double (fn)(double))
{
  double centerY = GetWindowHeight()/2.0;
  MovePen(start, centerY + fn(start));
  for (double x = start; x <= stop; x += Incr)
    LineTo(x, centerY + fn(x));
}
```

◊ Using function as data!
  ▪ Client passes function by name to `Plot` which graphs it

```c
int main()
{
  Plot(0, 2, sin);
  Plot(1, 10, sqrt);
  Plot(2, 5, MyFunction);
  Plot(2, 5, GetLine); // doesn't compile!
  ...
}
```

Back to Set

◊ Set needs to compare elements to establish order
◊ Default strategy applies relational ops:
  ```c
  {
    if (one == two) return 0;
    else if (one < two) return -1;
    else return 1;
  }
  ```
◊ What happens if this doesn't make sense for the client's type?
  ▪ E.g. == and < don't work on this type
Templated compilation error

```cpp
struct studentT {
    string first, last;
    int idNum;
    string emailAddress;
};

int main()
{
    Set<studentT> students;

    // Generates a compile error when instantiating the template:
    Error: no match for 'operator==' in 'one == two'
    Error   : illegal operands 'studentT' == 'studentT'
                (point of instantiation: 'main()')
    cmpfn.h line 25       if (one == two) return 0;

    // This is because < and == don't work for structs!
```

Suppling callback function

```cpp
struct studentT {
    string first, last;
    int idNum;
};

int CmpById(studentT a, studentT b)
{
    if (a.idNum < b.idNum) return -1;
    else if (a.idNum == b.idNum) return 0;
    else return 1;
}

int main()
{
    Set<studentT> set(CmpById); // ok!
```

Client callback function

- Functions as data provides solution!
  - Set written to use a function to compares two elements
  - By default it uses OperatorCmp, which applies <, ==
- Client can supply their own function
  - Must match prototype as specified by Set
  - Takes two elements, returns int
- Client's function does comparison of elements
  - Using desired info to get right sense of equal/order
  - Result is negative/zero/positive
- Client passes function to Set constructor
  - Set holds onto fn, and will callback client whenever it needs to compare two elements

Building things: ADTs rock!

- Map of Set
  - Google's web index (word to matching pages)
- Vector of Queues
  - Grocery store checkout lines
- Set of sets
  - Menu for a smoothie shop
- Stack of Maps
  - Compiler stores local variables and enter/exit nested scopes
Solving problems recursively

- Recursion is an indispensable tool in a programmer’s toolkit
  - Simple solutions to complex problems
  - Elegance can lead to better programs: easier to modify, extend, verify
- Get help solving the problem from coworkers (clones) who work and act like you do
  - Delegate similar, smaller problem to clone
  - Combine result from clone(s) to solve total problem

Recursive decomposition

- Standard decomp divides problem into dissimilar subproblems
  - Read file, store numbers, sort, ...
- Recursive decomp divides problem into smaller versions of same problem
  - Campus survey
  - Phone trees
  - Fractal drawing
- Recursive problems have "self-similar" structure in solution