Admin

- Sections start this week
  - Section assignments e-mailed, revisit signup page to switch
- Compiler installation fun
  - Any news will post to announcements on class web site
- Today's topics
  - C++ stream classes
  - CS106 class library: Scanner, Vector
- Reading
  - Reader Ch. 3, Handout 14 (today & next)

Lecture #4

C++ console I/O

- Stream objects cout/cin
  - cout is the console output stream, cin for console input
  - << is stream insertion, >> is stream extraction

```
#include <iostream>

int main()
{
      int x,y;
      cout << "Enter two numbers: ";
      cin >> x  >> y;
      cout << "You said: " << x << " and " << y << endl;
```

- Safer, easier read from console using our simpio.h

```
#include "simpio.h"

int main()
{
   int x = GetInteger();
   string answer = GetLine();
```

C++ file I/O

- File streams declared in <fstream>
  - streams are objects, dot notation used
  - ifstream for reading, ofstream for writing

```
#include <fstream>

ifstream in;
ofstream out;
```

- Use open to attach stream to file on disk

```
in.open("names.txt");
out.open(filename.c_str()); // requires C-string!
```

- Check status with fail, clear to reset after error

```
if (in.fail())
    in.clear();
```

Stream operations

- Read/write single characters

```
ch = in.get();
out.put(ch);
```

- Read/write entire lines

```
getline(in, line);
out << line << endl;
```

- Formatted read/write

```
in >> num >> str;
out << num << str;
```

- Use fail to check for error

```
if (in.fail()) ...
```

Class libraries

- Some libraries provide free functions
  - `RandomInteger`, `getline`, `sqrt` etc
- Other libraries provide classes
  - `string`, `stream`
- Class = data + operations
  - Tight coupling between value and operations that manipulate it
  - Class interface describes *abstraction*
    - Models string/time/ballot/database/etc with appropriate features
- Client use of object
  - Learn the abstraction, use public interface
  - Unconcerned with implementation details

Why is OO so successful?

- Tames complexity
  - Large programs become interacting objects
  - Each class developed/tested independently
  - Clean separation between client & implementer
- Objects can model real-word
  - Time, Ballot, ClassList, etc
  - Build on existing understanding of concepts
- Facilitates re-use
  - Also easily change/extend class in future

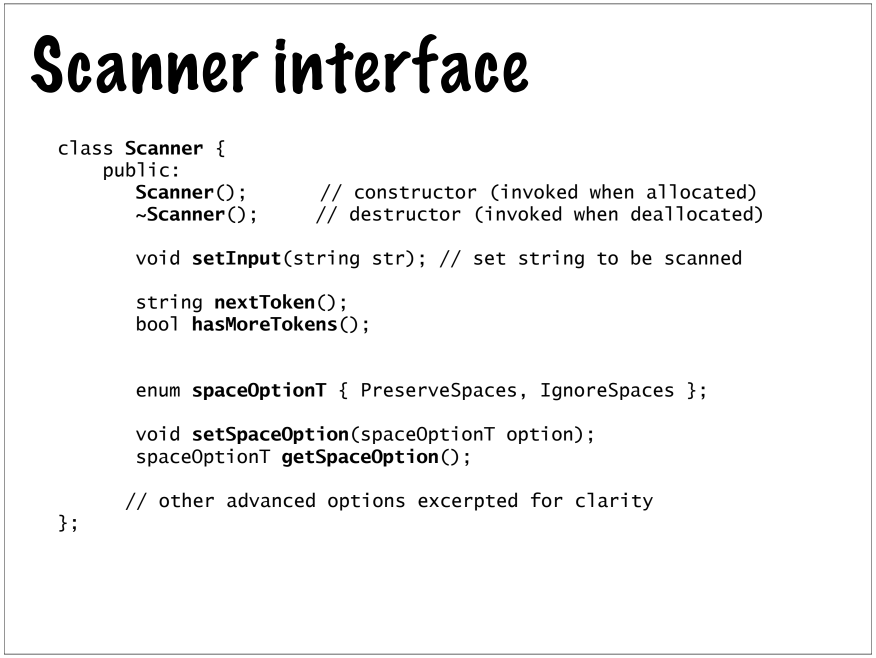
CS106 class library

- Provide common functionality, highly leveraged
  - Scanner
  - Vector, Grid, Stack, Queue, Map, Set
- Why?
  - Living "higher on the food chain"
  - Efficient, debugged
  - Clean abstraction
- We study as client and later as implementer
  - Why client-first?

CS106 Scanner

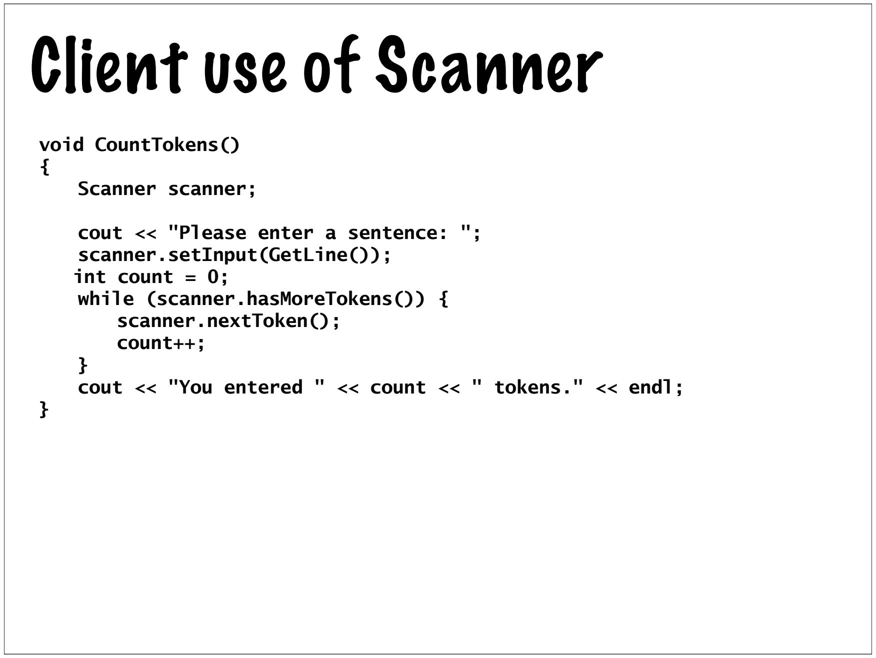
- Scanner's job: break apart input string into tokens
  - Mostly divide on white-space
  - Some logic for recognizing numbers, punctuation, etc.
- Operations
  - `setInput`
  - `nextToken/hasMoreTokens`
  - Fancy options available with set/get
- Used for?
  - Handling user input, reading text files, parsing expressions, processing commands, etc.

This □ **line** □ **contains** □ **10** □ **tokens** □ **.**



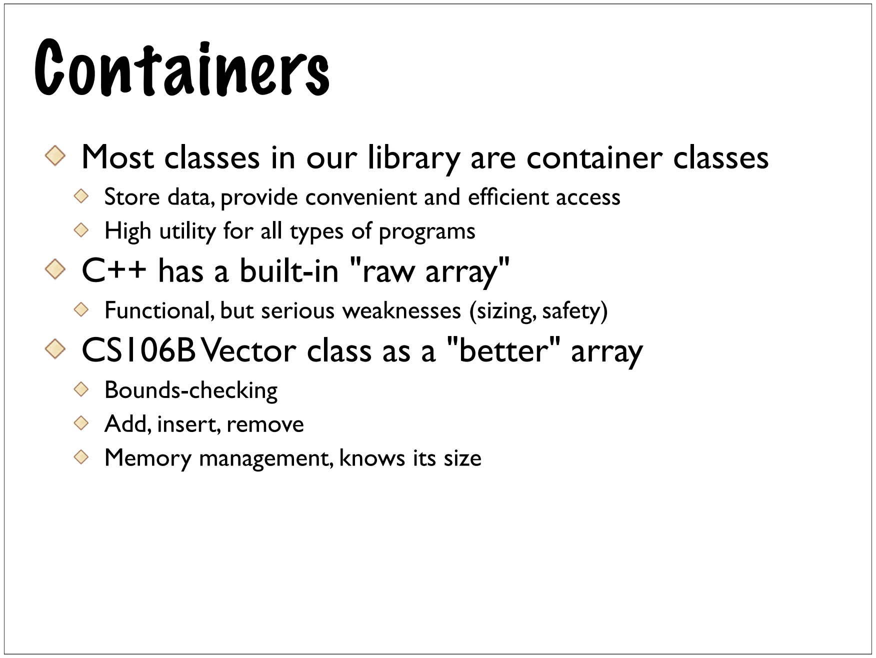
Scanner interface

```
class Scanner {
    public:
       Scanner();      // constructor (invoked when allocated)
       ~Scanner();     // destructor (invoked when deallocated)

       void setInput(string str); // set string to be scanned

       string nextToken();
       bool hasMoreTokens();


       enum spaceOptionT { PreserveSpaces, IgnoreSpaces };

       void setSpaceOption(spaceOptionT option);
       spaceOptionT getSpaceOption();

      // other advanced options excerpted for clarity
};
```



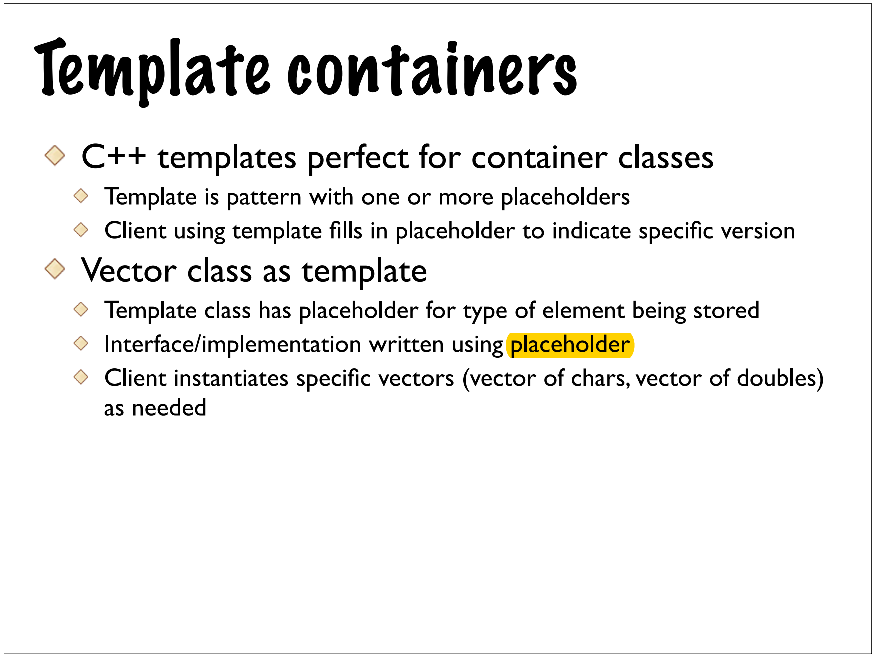
Client use of Scanner

```
void CountTokens()
{
    Scanner scanner;

    cout << "Please enter a sentence: ";
    scanner.setInput(GetLine());
    int count = 0;
    while (scanner.hasMoreTokens()) {
        scanner.nextToken();
        count++;
    }
    cout << "You entered " << count << " tokens." << endl;
}
```



Containers

- Most classes in our library are container classes
  - Store data, provide convenient and efficient access
  - High utility for all types of programs
- C++ has a built-in "raw array"
  - Functional, but serious weaknesses (sizing, safety)
- CS106B Vector class as a "better" array
  - Bounds-checking
  - Add, insert, remove
  - Memory management, knows its size



Template containers

- C++ templates perfect for container classes
  - Template is pattern with one or more placeholders
  - Client using template fills in placeholder to indicate specific version
- Vector class as template
  - Template class has placeholder for type of element being stored
  - Interface/implementation written using placeholder
  - Client instantiates specific vectors (vector of chars, vector of doubles) as needed

Vector interface

```
template <typename ElemType>
 class Vector {

  public:
    Vector();
     ~Vector();

    int size();
    bool isEmpty();

    ElemType getAt(int index);
    void setAt(int index, ElemType value);

    void add(ElemType value);
    void insertAt(int pos, ElemType value);
    void removeAt(int pos);
};
```

Templates are type-safe!

```
#include "vector.h"

void TestVector()
{
    Vector<int> nums;
   nums.add(7);

   Vector<string> words;
   words.add("apple");

   nums.add("banana");          // COMPILE ERROR!
   char c = words.getAt(0);     // COMPILE ERROR!
   Vector<double> s = nums;     // COMPILE ERROR!
}
```

Rules for template clients

- Client includes interface file as usual
  - `#include "vector.h"`
- Client must specialize to fill in the placeholder
  - Cannot use Vector without qualification, must be `Vector<char>`, `Vector<locationT>` , ...
  - Applies to declarations (variables, parameters, return types) and calling constructor
- Vector is specialized for its element type
  - Attempt to add `locationT` into `Vector<char>` will not compile!

Client use of Vector

```
#include "vector.h"

Vector<int> MakeRandomVector(int sz)
{
   Vector<int> numbers;
   for (int i = 0; i < sz; i++)
      numbers.add(RandomInteger(1, 100));
   return numbers;
}

void PrintVector(Vector<int> &v)
{
   for (int i = 0; i < v.size(); i++)
      cout << v[i] << " ";
}

int main()
{
   Vector<int> nums = MakeRandomVector(10);
   PrintVector(nums);
   ...
```