~~Admin~~

- ~~Section signups available on web, now until Sun 5pm~~
- ~~CS and the Honor Code~~
- ~~Alternate final exam~~
  - ~~I relented. Will offer final Th Mar 20 12:15-3:15pm. Absolutely NO other alternates.~~
- ~~Cafe hangout today after class in Terman — join us!~~
- ~~Today's topics~~
  - ~~C++ syntax and structure, procedural paradigm~~
  - ~~User-defined types, parameter passing~~
- ~~Reading~~
  - ~~Handout 4, Reader Ch. 1, 2.1, 2.6 (today)~~
  - ~~Ch. 3(next)~~

C++ vs Java: what's the same?

- General syntax
  - comment sequence
  - use of braces, parentheses, commas, semi-colons
  - variable/parameter declarations, function call
- Primitive variable types
  - char, int, double, but note Java boolean is C++ bool
- Operators
  - arithmetic, relational, logical
- Control structures
  - for, while, if/else, switch, return

Dissecting a C++ program

```
/*
 * average.cpp
 * ------------
 * This program adds scores and prints their average.
 */

#include "genlib.h"
#include "simpio.h"
#include <iostream>

const int NumScores = 4;

double GetScoresAndAverage(int numScores);

int main()
{
   cout << "This program averages " << NumScores << " scores." << endl;
   double average = GetScoresAndAverage(NumScores);
   cout << "The average is " << average << "." << endl;
   return 0;
}
```

average.cpp (cont'd)

```
/* Function: GetScoresAndAverage
 * Usage: avg = GetScoresAndAverage(10);
 * -------------------------------------
 * This function prompts the user for a set of values and returns
 * the average.
 */
double GetScoresAndAverage(int numScores)
{
    int sum = 0;
    for (int i = 0; i < numScores; i++) {
        cout << "Next score? ";
        int nextScore = GetInteger();
        sum += nextScore;
    }
    return double(sum)/numScores;
}
```

C++ user-defined types

- Enumerations
  - Define new type with set of constrained options

```
enum directionT {North, South, East, West};

directionT dir = East;
if (dir == West) ...
```

- Records
  - Define new type which aggregates a set of fields

```
struct pointT {
  double x;
  double y;
};

pointT p, q;
p.x = 0;
p = q;
```

C++ parameter passing

- Default is *pass-by-value*
  - Parameter copies value, changes affect local copy

```
void Binky(int x, int y)
{
  x *= 2;
  y = 0;
}
```

```
int main()
{
   int a = 4, b = 20;
   Binky(a, b);
   ...
```

- Add & to declaration for *pass-by-reference*
  - Parameter is now reference to original variable, which can change

```
void Binky(int &x, int y)
{
  x *= 2;
  y = 0;
}
```

```
int main()
{
   int a = 4, b = 20;
   Binky(a, b);
   ...
```

  - Ref param also used for efficiency to avoid copying large data

C++ libraries

- Groups related operations
  - Header file provides function prototypes and usage comments
  - Compiled library contains implementation
- C++ standard libraries
  - e.g. string, iostream, fstream
  - `#include <iostream>`
  - Terse, lowercase names: `cout getline substr`
- CS106 libraries
  - e.g. simpio, random, graphics
  - `#include "random.h"`
  - Capitalized verbose names: `GetInteger RandomChange DrawLine`

Admin

- Sections meet this week
  - Section assignments e-mailed to you tomorrow
- Assign 1 out
- Handouts 5 & 7 come in two flavors
  - Take ONE version (Mac or Windows) depending your platform
- Today's topics
  - Libraries, C++ string and stream classes
- Reading
  - Handout 4, Reader Ch. 3 (today & next)

C++ libraries

- Groups related operations
  - Header file provides function prototypes and usage comments
  - Compiled library contains implementation
- C++ standard libraries
  - e.g. string, iostream, fstream
  - `#include <iostream>`
  - Terse, lowercase names: `cout` `getline` `substr`
- CS106 libraries
  - e.g. simpio, random, graphics
  - `#include "random.h"`
  - Capitalized verbose names: `GetInteger` `RandomChance` `DrawLine`

CS106 random.h

- Library of functions to provide randomness
  - Support for shuffling, dice-rolling, coin-flipping, etc.
  - Free functions
- `void Randomize()`
  - Call once at start to initialize new random sequence
- `int RandomInteger(int low, int high)`
  - Returns int chosen from at random from range low-high inclusive
- `double RandomReal(double low, double high)`
  - Same, but for real values
- `bool RandomChance(double probability)`
  - Returns true with odds of probability, false otherwise
- Coherent, convenient, complete

C++ string

- Models a sequence of characters
- string defines a class, strings are objects
  - many operations are member functions that operate on receiver string
- Simple operations
  - member function .length returns number of chars
  - square brackets to access individual chars
  - C++ strings are mutable! (unlike Java)

```
int main()
{
  string s;

  s = "cs106";
  for (int i = 0; i < s.length(); i++)
     s[i] = toupper(s[i]);
```

Operators on strings

- Assign using =, makes new copy
- Compare with relational ops (<, ==, >=, ...)
  - lexicographic ordering
- + is overloaded to do concatenation
  - operands must be chars or strings only

```
int main()
{
  string s, t = "hello";

  s = t;
  t[0] = 'j';
  s = s + ' ';
  if (s != t)
      t += t;
```

string member functions

- Invoke member functions using dot notation

  `str.function(args)`
- Sample member functions:

  `int length()`

  `int find(char ch, int pos = 0)`

  `int find(string pattern, int pos = 0)`
  - returns index of first occurrence or string::npos if not found

  `string substr(int pos, int len)`
  - returns new string, copies len characters starting from pos

  `void insert(int pos, string txt)`
  - changes receiver, inserts txt at pos

  `void replace(int pos, int len, string txt)`
  - changes receiver, removes len chars start at pos, replace with txt

  `void erase(int pos, int len)`
  - changes receiver, removes len chars starting at pos

CS106 strutils.h

- Few convenience free functions for string
- Converting between case
  - `string ConvertToLowerCase(string s)`
    `string ConvertToUpperCase(string s)`
- Converting numbers to string and back
  - `int StringToInteger(string s)`
    `string IntegerToString(int num)`

    `double StringToReal(string s)`
    `string RealToString(double num)`

C++ string vs C-string

- C++ inherits legacy of old-style C-string
  - (pointer to character array, null-terminated)
  - String literals are actually C-strings
- Converting C-string to C++ string
  - Happens automatically in most contexts
  - Can force using `string("abc")`
- Converting C++ string to C-string
  - Using member function `a.c_str()`
- Why do you care?
  - Some older functionality requires use of C-string
  - C-string not quite compatible with concatenation

Concatenation pitfall

- If one operand is true C++ string, all good

```
string str = "abc";
str + "def";
str + 'd';
str + ch;
```

- If both operands are C-string/char, bad times

```
"abc" + "def";     // won't compile
"abc" + 'd';      // compiles, but doesn't work
"abc" + ch;       //  same
```

- Can force conversion if needed

```
string("abc") + ch;
```

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();
```

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)

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?

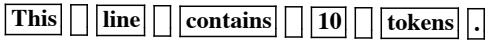
- Tames complexity
  - Large programs become interacting objects
  - Each class developed/tested independently
  - Clean separation between client & implementer
- Objects can model real-word
  - Time, Ballot, ClassList, etc
  - Build on existing understanding of concepts
- Facilitates re-use
  - Also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



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);
   ...
```

Admin

- Assign 1 due next Wed
  - Web announcements for late-breaking news
- MLK, Jr Day on Monday, no lecture
- Today's topics
  - CS106 class library: Vector, Grid, Stack, Queue
- Reading
  - Handout 14 (today & next)
- A note about arrays/pointers
  - Covered in Ch. 2, but we wait to introduce until we have a good use for them, so don't worry for now
- Terman cafe today after lecture

Client use of templates

- 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!

Vector class

- Indexed, linear homogenous collection
  - Knows its size
  - Access is bounds-checked
  - Storage automatically handled (grow & shrink)
  - Convenient insert/remove
  - Deep-copy on assignment, pass/return-by-value
- Usage
  - Constructor creates empty vector
  - Add/insert adds new element
  - Access elements using setAt, getAt or operator []
- Useful for:
  - every kind of list you can imagine!

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);
};
```

Template specialization

```
class Vector<double> {

 public:
  Vector<double>();
   ~Vector<double>();

  int size();
  bool isEmpty();

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

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

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);
   ...
```

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!
}
```

Grid class

- 2-D homogenous collection indexed by row & col
  - Access to elements is bounds-checked
  - Deep-copy on assignment, pass/return by value
- Usage
  - Set dimensions in constructor (can later resize)
  - Elements have default value for type before explicitly assigned
  - Access elements using getAt/setAt or operator ()
- Useful for:
  - Game board
  - Images
  - Matrices
  - Tables

Grid interface

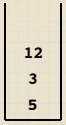
```
template <typename ElemType>
 class Grid {

   public:
    Grid();
     Grid(int numRows, int numCols); // overloaded constructor
     ~Grid();

    int numRows();
    int numCols();

    ElemType getAt(int row, int col);
    void setAt(int row, int col, ElemType value);

    void resize(int numRows, int numCols);
};
```

Client use of Grid

```
#include "grid.h"

// Returns a new 3x3 grid of chars, where each
// elem is initialized to space character
Grid<char> CreateEmptyBoard()
{
   Grid<char> board(3, 3); // create 3x3 board of chars

   for (int row = 0; row < board.numRows(); row++)
      for (int col = 0; col < board.numCols(); col++)
         board(row, col) = ' '; // board.setAt(row, col, ' ')

   return board; // btw, it's ok to return object
}
```

Stack class

- Linear collection, last-in-first-out
  - Limited-access vector
  - Can only add/remove from top of stack
  - Deep-copy on assignment, pass/return by value
- Usage
  - Constructor creates empty stack
  - push to add objects, pop to remove
- Useful for:
  - Reversing a sequence
  - Managing a series of undoable actions
  - Tracking history when web browsing



Stack interface

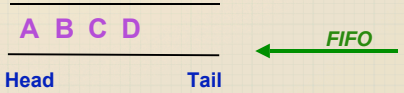
```
template <typename ElemType>
  class Stack {

   public:
     Stack();
     ~Stack();

     int size();
     bool isEmpty();

     void push(ElemType element);
     ElemType pop();
     ElemType peek();
};
```

Client use of Stack

```
void ReverseResponse()
{
    cout << "What say you? ";
    string response = GetLine();

    Stack<char> stack;
    for (int i = 0; i < response.length(); i++)
        stack.push(response[i]);

    cout << "That backwards is :";
    while (!stack.isEmpty())
        cout << stack.pop();
}
```

Queue class

- Linear collection, first-in-first-out
  - Limited-access vector
  - Can only add to back, remove from front
  - Deep-copy on assignment, pass/return by value
- Usage
  - Constructor creates empty queue
  - enqueue to add objects, dequeue to remove
- Useful for:
  - Modeling a waiting line
  - Storing user keystrokes
  - Ordering jobs for a printer
  - Implementing breadth-first search



Queue interface

```
template <typename ElemType>
  class Queue {

    public:
      Queue();
      ~Queue();

      int size();
      bool isEmpty();

      void enqueue(ElemType element);
      ElemType dequeue();
      ElemType peek();
   };
```

Client use of Queue

```
void ManageQueue()
{
   Queue<string> queue;

   while (true) {
      cout << "? ";
      string response = GetLine();
      if (response == "") break;
      if (response == "next") {
         if (queue.isEmpty())
           cout << "No one waiting!" << endl;
         else
           cout << "Handle" << queue.dequeue() << endl;
      } else {
         queue.enqueue(response);
         cout << "Add" << response << endl;
      }
   }
}
```

Nested templates

- Queue can hold stacks or vector of vector, etc

```
Vector<Queue<string> > checkoutLines;
Grid<Stack<string> > game;
```

- Need space between >> closers
  - Otherwise compiler see stream extraction
- Can use typedef to make shorthand name
  - `typedef Vector<Vector<int> > calendarT;`