ENERGY 211 / CME 211
Computer Programming in
C++ for Earth Scientists and
Engineers

Lecture 1
September 22, 2008

Why This Course?

• To help students improve or acquire the
programming skills they need for
research in earth sciences or
engineering
• Alternative to CS106X, with more
relevant examples and exercises
What is Covered?

• Learning C++
  – Basics
  – Classes
  – Standard C++ library, including STL

• Software Engineering Practices
  – Computer architecture essentials
  – Design, testing, debugging
  – Efficient memory usage

• Special Topics

Course Staff

• Instructor: Prof. Jim Lambers
  – Acting assistant professor, ERE/ICME
  – Former software engineer
  – Email: lambers@stanford.edu

• TA: Alejandro David Leiva
  – MS student, ERE
  – Email: aleiva@stanford.edu

• TA: Yifan Zhou
  – MS student, ERE
  – Email: yifanz@stanford.edu
Course Web Sites

• http://energy211.stanford.edu
  – Primary site, for easier access to course materials and information
• http://coursework.stanford.edu
  – For viewing grades, receiving email announcements
• Lecture notes, assignments available at either site

Your Grade

• No midterm or final exam
• 6 programming projects, 10-15% each
• Final project, 30%
• Bonus: 1% added to overall grade if at least 5 coding-related questions asked by e-mail to instructor
• Letter grades: straight scale (or nicer)
Programming Projects

• 6 projects, one week each
• Can (and should!) work with a partner
  – Can be different partner for each project
  – No changing partners in the middle of a project
• Must be submitted electronically (details to come)
• First project to be distributed on Friday, October 3

Late Days

• Programming projects must be submitted electronically by 11:59pm on the due date
• You have five late days for the entire quarter that may be used at any time, no questions asked
• Use them wisely, they might be all you get!
Why C++?

- To maximize computational efficiency, best to use low-level programming languages (easy to translate into efficient machine code)
- To efficiently build large applications, best to use high-level languages that support modularity, abstraction and encapsulation
- C++ combines aspects of both

Low-level C/C++

```c
#include <stdlib.h>

extern ssize_t write(int, const void *, size_t);

int main()
{
    write(1, "Hello world!\n", 13);
}
```

```assembly
0:  1b 00 00 00 sethi %hi(0x0), %o5
4:  82 10 00 0f mov  %o7, %g1
8:  92 03 60 00 add  %o5, 0, %o1
c:  90 10 20 01 mov  1, %o0
t:  94 10 20 0d mov  13, %o2
14: 40 00 00 00 call 0x14
18: 9e 10 00 01 mov  %g1, %o7
```
High-level C++

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello world!" << std::endl;
}
```

- Easier to understand this program’s purpose: to send text and end-of-line to output device
- No need for device descriptors, computing text length, or escape sequences
- Being programmer-friendlier carries a cost: assembly code is nearly 4 times as long!

A Few Buzzwords

- Modularity: the extent to which a program is decomposed into separate modules, thus simplifying design
- Abstraction: the practice of reducing or factoring out details so that one can focus on only a few concepts at a time
- Encapsulation: hiding the design of a task from the rest of the application, thus protecting it from changes
What C++ Offers

• All of the power and efficiency of C (which is a subset)
• Object-oriented programming (though not truly an object-oriented language)
• Generic programming, through templates
• Definition and extension of operators
• Freedom to use paradigm of choice

Why Not Just Use MATLAB?

• MATLAB is useful for proof-of-concept, but not for release-grade software
• Its language is interpreted, not compiled into machine code
• Only offers a subset of the functionality of a compiled language such as C++
• This lack of flexibility significantly, and adversely, impacts usability and performance
Examples of Drawbacks

- MATLAB uses LAPACK and BLAS for its low-level linear algebra operations
- Use of other libraries is much more difficult and less efficient
- Forced to use MATLAB’s data structures for dense and sparse matrices, or other data
- Lack of control over how data is accessed results in needless copying

Effective Software Engineering

- Involves much more than just designing and implementing algorithms
- Criteria for a program’s usefulness:
  - Does it accomplish its intended task, and if so, with what limitations?
  - How efficient is it, in the given computing environment?
  - How robust is it? Does it handle adverse conditions gracefully?
Hard Lessons

• The sooner you start writing code, the longer it takes you to finish
• Untested code will almost always fail
• What works today can be broken tomorrow
• The first person to use your code will input the one case that it doesn’t handle
• Document code well, because your code can be unintelligible, even to you

Sound Programming Practices are Essential!

• In 1991 Gulf War, US Patriot missiles intercepted Iraqi SCUD missiles by estimating velocity = distance / time
• Time interval obtained by subtracting times at which position was determined
• Times measured as seconds since tracking devices first activated in the field
Tracking SCUD Missiles

Correct

Incorrect

Source: U.S. GAO Report

Little Details Aren’t Really Little

- Time values large, but close together, so subtraction inaccurate, due to *catastrophic cancellation*
- Velocity estimates useless, so missile’s position could not be determined
- Interception failed, led to 28 deaths
- What is a simple remedy?
Next Time

• Overview of C++:
  – History
  – Design
• How to write and run C++ programs
  – Windows
  – UNIX/LINUX
• Understanding simple C++ programs