Welcome to CSC340: Programming Methodology.

A brief course introduction.

Hui Yang
Computer Science Department
San Francisco State University
http://www.cs.sfsu.edu/~huiyang/

Learning Goals

- Programming in C++ to solve problems
  - Basic and advanced topics
- Recursively solve a set of problems
- Algorithm efficiency analysis
  - Nested for-loops
  - Recursive algorithms
- Tentative topics: graph algorithms and binary search trees
- **A tentative weekly schedule**

Why the 35-year-old C++ still dominates 'real' dev?

- **A recent interview with Bjarne Stroustrup by InfoWorld:**
  - http://www.infoworld.com/t/application-development/stroustrup-why-the-35-year-old-c-still-dominates-real-dev-248457 (August 15, 2014)
- **Synopsis**
  - [C++] age remains vital and relevant ... because **of its ability to handle complexity (**"nothing that can handle complexity runs as fast as C++"), making it the **go-to solution** for telecom, financial, and embedded applications and online systems such as Amazon and Google.
  - he used C++ for projects that "required a real programming language and **real performance**" [as C++ is for] high performance, high reliability, small footprint, low energy consumption, all of these good things.
  - it's always been used together with some scripting language or other.
  - C++ is **NOT FOR** small apps or hobbyists

If C++ or Java were vehicles ...

- **C is** the great all-arounder: compact, powerful, goes everywhere, and reliable in situations where your life depends on it.
- **C++ is** the new C — twice the power, twice the size, works in hostile environments, and if you try to use it without care and special training you will probably crash.
- **Java is** another attempt to improve on C. It sort of gets the job done, but it's way slower, bulkier, spews pollution everywhere, and people will think you're a redneck.
- More at: http://crashworks.org/if_programming_languages_were_vehicles/?utm_content=buffere2b6d&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer

Housekeeping Items

- Instructor & TA
- Website
- Textbooks
- Prerequisites
- Grading
- Policies on plagiarism and course repeat
- Primary teaching & learning tools
- Your to-do list

Your Human Contacts

- **Instructor**
  - Hui Yang, Ph.D. (huiyang_at_sfsu.edu)
  - Office: 966 Thornton Hall
  - Office hours: 4:00—4:45pm, Tuesday/Thursday (or by appointment)
- **Graders**
  1. Anthony
  2. Kevin
  3. Maya
  4. Xuejing
- **Graders' office hours**
  - To be announced.

Course Website

- **The website for this course is**
  - **http://ilearn.sfsu.edu/**
  - Nearly all course materials such as lecture slides, in-class demo codes, and assignments will be posted on this website.
- **Discussion forum**
  - Not a substitute for in class participation
  - A great venue for asking and finding answers to many questions.
  - Use it well.
- **Checking the site is *your* responsibility.**
  - I will send you reminders on important deadlines, such as homework, exams, etc.

Textbooks

- **Textbook**
  - Data abstraction & problem solving with C++, by Frank M. Carrano, Sixth edition, Addison Wesley
- **Recommended books**
  - *C++ for Java Programmers (Paperback)*, by Mark Allen Weiss , Prentice Hall.
  - Experienced C++ user
    - *The C++ Programming Language (any edition), by* Bjarne Stroustrup,
  - C++ beginners
    - *Problem solving with C++*, 7th edition, by Walter Savitch,
- **Supplementary materials**
  - Will be posted to iLearn as needed

Prerequisites of CSC340

- **CSC340 is a required course for computer science majors.**
- **Recommended study sequence**
  - (csc220, csc230)→csc340→csc413→csc510
  - http://cs.sfsu.edu/undergrad/under-rec-sequence.html
- **Prerequisites: A grade of C or better in**
  - Csc220: Data structures
  - Csc230: Discrete mathematical structures for computer science
- **This is a firm requirement.**

You are required to have mastered:

- **Object-oriented design and programming**
- **Programming in Java**
  - E.g., flow control, variables, file I/Os, and arrays
- **Fundamental data structures and algorithms**
  - Stacks, queues
  - Tables
  - Sets
  - Maps
  - Basic hash tables
  - Basic sorting and searching algorithms
- **How to compute the sum of an arithmetic series**
- **How a loop, especially a nested loop, works**
- **Recursively define and solve a problem**

Assignments: submission and grading

- **Two docs on iLearn**
  - What to turn in?
  - Grading a project
- **Late submission policy**

Time line	Penalty
Within **the first 24 hours** since the due time	15 points
Within **the second 24 hours** since the due time	30 points
Within **the third 24 hours** since the due time	45 points
Beyond 72 hours	Not Accepted.

Everything you do counts.

- **Class and online participation** **5% (2.5+2.5)**
- **Quizzes (~3)** **3%**
- **Assignments (~7)** **37%**
- **Mid term exam** **25%**
- **Final exam** **30%**
- **Bonus points**
  - Optional assignments might be assigned for everyone: 2-3 bonus points
  - Up to 2 bonus points for overall improvement in the final exam.
  - Advanced C++ users:
    - Work with the instructor to define a medium-sized project that addresses real-world problems.
    - Earn up to 3 bonus points.

Letter Grades: Mapping Schema

Weighted Score	Grade
[90, 100]	A
[86, 89.99]	A-
[82, 85.99]	B+
[78, 81.99]	B
[74, 77.99]	B-
[71, 73.99]	C+
[67, 70.99]	C
[63, 66.99]	C-
[59, 62.99]	D+
[55, 58.99]	D
[51, 54.99]	D-
[0, 50.99]	F

Mandatory Policies: Plagiarism

- **Absolutely zero tolerance.**
- **Plagiarism**: please carefuly read the department policy http://cs.sfsu.edu/plagarism.html
- **Excerpt**

  "While we encourage students to work together to learn and understand concepts and material related to homework and other graded work, unless permission from instructors is given, students are NOT to share their specific homework solutions or any work (e.g. math formulas, code, reports etc, in hard copy or in electronic form) with other students, especially before submission deadlines for these assignments. Note that sharing in electronic form makes intentional or unintentional plagiarism very easy.

  If plagiarism occurs, both the students sharing and those using the work as their own are considered to be responsible for plagiarism, and will be subject to appropriate penalties."

Mandatory Policies: More.

- Generally, there will be no **make-up exams** and no **incomplete grades** given. If you have to miss an exam, you must notify the instructor before the exam. If any of the scheduled exams dates are in conflict with your religious observances, you must notify your instructor, in writing, during the first two weeks of the semester. If you have an acceptable, documented excuse, you may be given a make-up exam or be given the average score of other exams at the discretion of the instructor. Otherwise, a missed exam is graded zero points.
- **Drop and withdrawal** are students' responsibilities.
- **Course repeat policy**: you can repeat a class only once.

Primary Teaching/Learning Tools

- **In the classroom**
  - Slides
  - Whiteboard
  - In class coding demonstration
  - Hands-on quizzes (not graded)
- **Outside the classroom**
  - Hands-on session supervised by a TA
  - Homework assignments and optional projects for advanced C++ programmers
  - iLearn discussion forum
  - Feedback from you

Feedback from successful students.

- **Have the right attitude and be passionate**
- **Keep in attendance, keep in track**
- **Read the slides & chapters before each class**
  - Class time is for review, clarification.
- **Spend at least four hours offline for every hour in class,**
- **Overcome coding anxiety; code as much as possible.**
  - CSC412 will help.
- **Take homework seriously. Work on it independently.**
  - Learn to debug using a debugging tool.
- **Review slides/notes before an exam.**
- **Hire a personal tutor and use the book for beginners if necessary.**
- **Ask for help: going to instructor & TA's office hours**
- **Don't bet everything on the final exam**
- **Pay attention to the extra credit homework**
- **Backup your work regularly!**

Roll-call and Waitlisting

- **Roll-call**
- **Waitlisting**
  - Put down the following information:
    - name, email, SFSU ID, class level, CSC220, CSC230.
  - Priority order
    - Conditional grads
    - Students with the required prerequisites
    - Other compelling reasons: case-by-case
  - Will issue a permit once seats are available.

Development Environments

- **IDE-Integrated Development Environment (Preferred)**
  - Microsoft Visual Studio
  - Mac OS: Xcode
  - Open Source: Dev-C++
    - Windows XP: Dev-C++ 4.9.9.2
    - Windows 8: Orwell Dev-C++ 5.4.2
      - http://orwelldevcpp.blogspot.com/
      - Download *"The setup which includes TDM-GCC x64 4.7.1 can be downloaded here (44MB)"*
- **Command line**
  - Linux, Unix

Demonstration

- **Coding a simple C++ program using Dev-C++.**
- **The small world of doodlebugs and ants.**
  - Separate compilation
  - C++ ADTs
  - Inheritance
  - Polymorphism
  - Pointers

The goal for this programming project is to create a simple two-dimensional predator-prey simulation. In this simulation the prey are ants and the predators are doodlebugs. These critters live in a world composed of a 20×20 grid of cells. Only one critter may occupy a cell at a time. The grid is enclosed, so a critter is not allowed to move off the edges of the world. Time is simulated in time steps. Each critter performs some action every time step.

The ants behave according to the following model:

- Move. Every time step, randomly try to move up, down, left, or right. If the neighboring cell in the selected direction is occupied or would move the ant off the grid, then the ant stays in the current cell.
- Breed. If an ant survives for three time steps, then at the end of the time step (that is; after moving) the ant will breed. This is simulated by creating a new ant in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available, then no breeding occurs. Once an offspring is produced an ant cannot produce an offspring until three more time steps have elapsed.

The doodlebugs behave according to the following model:

- Move. Every time step, if there is an adjacent ant (up, down, left, or right), then the doodlebug will move to that cell and eat the ant. Otherwise, the doodlebug moves according to the same rules as the ant. Note that a doodlebug cannot eat other doodlebugs.
- Breed. If a doodlebug survives for eight time steps, then at the end of the time step it will spawn off a new doodlebug in the same manner as the ant.
- Starve. If a doodlebug has not eaten an ant within the last three time steps, then at the end of the third time step it will starve and die. The doodlebug should then be removed from the grid of cells.

During one turn, all the doodlebugs should move before the ants do.

To-Dos

- Review CSC220/230 materials (See slide #10)
- Read the syllabus carefully and report any of your time conflict (exam dates) by the end of the 4th week.
- Download and install either Viusal C++ or Dev-C++. URLs are available on iLearn.
- Read the following sections in your textbook
  - Required: Appendix k
  - Additional:
    - http://www.ittybittycomputers.com/Courses/Prior/ADS/C4Java.htm
    - http://web.engr.oregonstate.edu/~budd/Books/cforj/info/preface.pdf