Today

- Overview
- C++ Review

Assignments

- HW1 will be posted on Thursday
Syllabus Overview

Communication:

- Website: www.cs.gonzaga.edu/bowers/courses/cpsc223
- Piazza (via invite)
- GitHub classroom (requires GitHub account)
- Blackboard (for posting grades)

Getting Help:

- Piazza: Preferred for (basic) questions, clarifications, etc.
- Office hours: 11–12 Tu/Th; 11-12 & 1–2 Wed; or by appt

Grading:

- 60% homework
- 10% class participation
- 5% quizzes
- 25% exams (5% midterm, 7.5% final report, 12.5% final exam)

Software:

- Department VM and/or ada server
- cmake, make, google test (gtest), g++, gdb, valgrind, git, gnuplot
- editor of your choice (vs code, atom, emacs, vim, ...)

S. Bowers
Homework basics:

(Some) collaboration is encouraged ...

- can discuss and talk about assignments
- work together (as long as you are doing your own work)
- plagiarism not allowed: no splitting up work, no code sharing, etc.
- check with me if unsure

Late policy and resubmission policy

- 25% of max up to two weeks
- if score < 75%, can fix and resubmit (for max score of 75%)
- but note that homework assignments build on each other

Most CS jobs require you to demonstrate your ability to program

- questions drawn from much of what we discuss in 223!

Warning:

- longer, more complex programming than you may have seen so far
- class moves (fairly) quickly
- requires significant time commitment (start assignments early!)
- assignments like intense “exercises” (versus “real” programs)
Basic Terminology

A **Data Type** defines

- a set of **values** and **operations** over the values

A **Base** Data Type ... aka **Primitive** or **Atomic**

- is “built into” (provided by) the language
- e.g.: `int`, `double`, `bool`, `char`, sometimes `string`
- base-type values are "unstructured" (can’t be decomposed)

A **Composite** Data Type

- defines values that are structured (decomposable)
- e.g.: a record (struct) of base types, an array of integers

An **Abstract** Data Type (**ADT**)  

- is a data type without a specified implementation (usually many possible)
- usually for collections of data (e.g., instances can store 0 to \( n \) data items)
- e.g.: stack, queue, set, dictionary, tree, graph

A **Data Structure** (*emphasis of this class*)

- is a concrete implementation of a data collection
- prescribes a particular structure / organization for the collection
- e.g.: resizable array, linked list, hash table
- useful for implementing different ADTs with different performance properties
Survey and Course Topics

C++ Programming ... (first few weeks)
- but not a learn-to-program C++ class!
- we’ll cover a couple of new (minor) topics
- primarily more practice with classes, pointers, recursion, etc.

Different Data Structures ... (majority of class)
- Review of linked lists
- Resizable arrays
- Hash tables
- Binary search trees
- Plus more as time allows (e.g., heaps)

Data structures in the context of a few specific ADTs
- Sequences (first few assignments)
- Dictionary (most of semester) ... key-value pair collection
- Priority Queues (as time allows)

Algorithms and Analysis ... 
- Adding, removing, and searching for data (main focus)
- Sorting data ("classic" sorting algorithms) ... useful for searching
- Comparisons of algorithmic performance (complexity vs experimental)
**Basic C++ Review** *(some items that will come up)*

**Exercise Sheet:** Initialization, Pointers and Addresses, Signatures

**Advice:** ... from: *A Tour of C++* (Stroustrup)

1. Read the class style guide (available on course webpage)!
2. Minimize the scope of a variable
3. Prefer immutable data (e.g., via `const`)
4. Avoid uninitialized variables
5. Don’t declare a variable until you have a value to initialize it with
6. Keep use of pointers simple and straightforward
7. Use `nullptr` rather than 0 or `NULL`
8. State intent in comments