Welcome to CS106B!

• Today:
  • Course Overview
  • Where are We Going?
  • Introduction to C++
Course Staff

**Instructor:** Aubrey Gress  
(adgress@cs.stanford.edu)

**Head TA:** Michael Chang  
(mchang91@cs.stanford.edu)

The CS106B Section Leaders  
The CS106B Course Helpers
Course Website

http://cs106b.stanford.edu
Prerequisites

CS106A
(or equivalent)
Background Topics

- We assume you are familiar with:
  - Variables
  - Parameter passing
  - Functions
  - Classes and Objects
  - For/While Loops
  - If/else statements

- Okay if you need to do some background reading

- Most important thing is that you have some experience taking a problem and turning it into code
Required Reading

Programming Abstractions in C++
Required Reading

• Hard copies in the book store, electronic copy on the website.
  • Exams this quarter will not be open note (more on this in a couple slides).
  • You don't have to buy the hard copy, but it is highly recommended.
Grading Policies
Grading Policies

- 55% Assignments
- 20% First Midterm
- 20% Second Midterm
- 5% Section Participation
Grading Policies

- 55% Assignments
- 20% First Midterm
- 20% Second Midterm
- 5% Section Participation

Six Programming Assignments
Grading Policies

- 55% Assignments
- 20% First Midterm
- 20% Second Midterm
- 5% Section Participation
Grading Policies

First Midterm Exam

July 22\textsuperscript{nd}, 7-10pm
Grading Policies

Exam will **not** be written to take 3 hours. It will be written to take ~1 hour.

Exams are stressful and we want to eliminate at least one form of stress (the time component).

First Midterm Exam

**July 22**\(^{nd}\), **7-10pm**
Grading Policies

- 55% Assignments
- 20% First Midterm
- 20% Second Midterm
- 5% Section Participation
Grading Policies

- 55% Assignments
- 20% First Midterm
- 20% Second Midterm
- 5% Section Participation

Second Midterm Exam

August 12th, 7-10pm
Grading Policies

- 55% Assignments
- 20% First Midterm
- 20% Second Midterm
- 5% Section Participation
Exams

- Historically exams have been open note. This quarter the exam will **not** be open note
  - Rational: Allows us to ask simpler questions and ask more knowledge based questions.
  - Remember the course is curved.
- Before the first exam I'll cover strategies for studying the exam.
Discussion Sections

- Weekly discussion sections.
- Section attendance is **required** in CS106B.
- Sign up between Thursday, June 27 at 5PM and Sunday, June 30 at 5PM at [http://cs198.stanford.edu/section](http://cs198.stanford.edu/section)

- You don't need to (and shouldn't!) sign up for a section on Axess; everything is handled through the above link.
Discussion Sections

- Roughly ~10 students per section
- Get more experience using problem solving techniques from lecture
How Many Units?
int numUnits(bool isGrad, bool wantsFewerUnits) {

if (!isGrad)
    return 5;

if (!wantsFewerUnits)
    return 5;

if (reallyBusy()) {
    return 3;
}
else {
    return 4;
}
}
How Many Units?

```c
int numUnits(bool isGrad, bool wantsFewerUnits) {
    if (!isGrad) return 5;
    if (!wantsFewerUnits) return 5;
    if (reallyBusy()) {
        return 3;
    } else {
        return 4;
    }
}
```
How Many Units?

```c
int numUnits(bool isGrad, bool wantsFewerUnits) {
    if (!isGrad) return 5;
    if (!wantsFewerUnits) return 5;
    if (!reallyBusy()) return 4;
}
```
int numUnits(bool isGrad, bool wantsFewerUnits) {
    if (!isGrad) return 5;
    if (!wantsFewerUnits) return 5;
    if (reallyBusy()) {
        return 3;
    }
}

How Many Units?
int numUnits(bool isGrad, bool wantsFewerUnits) {
    if (!isGrad) return 5;
    if (!wantsFewerUnits) return 5;
    if (reallyBusy()) {
        return 3;
    } else {
        return 4;
    }
}
Getting Help
Getting Help

- LaIR Hours: Run by Section Leaders
  - Sunday – Wednesday, 7PM – 11PM
  - Starts next week.
  - Great time/place to work on assignments!
- Mike's Office Hours in Gates 160
  - Tuesday/Wednesday 3PM – 5PM
- Aubrey's Office Hours in Gates 160
  - Monday-Thursday 12PM - 1PM
  - Or by Appointment!
What's Next in Computer Science?
Goals for this Course

• Learn how to model and solve complex problems with computers.
Goals for this Course

- Learn how to model and solve complex problems with computers.

- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.
Goals for this Course

Learn how to model and solve complex problems with computers.

To that end:

- Explore common abstractions for representing problems.

Harness recursion and understand how to think about problems recursively.

Quantitatively analyze different approaches for solving problems.
CS106B totally rocks my socks.
Building a vocabulary of abstractions makes it possible to represent and solve a wider class of problems.
Goals for this Course

- **Learn how to model and solve complex problems with computers.**
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.
Goals for this Course

Learn how to model and solve complex problems with computers.

To that end:

- Explore common abstractions for representing problems.

  • Harness recursion and understand how to think about problems recursively.

Quantitatively analyze different approaches for solving problems.
Recursion: Fibonacci Numbers

- Fibonacci Numbers
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, ... 
  - Defined recursively:

\[
\text{fib}(n) = \begin{cases} 
  n & \text{if } n = 0 \text{ or } 1 \\
  \text{fib}(n-1) + \text{fib}(n-2) & \text{otherwise}
\end{cases}
\]

- What would this look like in code?
Recursion: Fibonacci Numbers

- Fibonacci Numbers
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
  - Defined recursively:

\[
fib(n) = \begin{cases} 
  n & \text{if } n = 0 \text{ or } 1 \\
  fib(n-1) + fib(n-2) & \text{otherwise}
\end{cases}
\]

- What would this look like in code?
- It's okay if this is hard to think about! It is for most people when they see it for the first (and second and third) time.
A *recursive solution* is a solution that is defined in terms of “smaller” instances of itself.
Thinking recursively allows you to solve an enormous class of problems cleanly and concisely.
Goals for this Course

- **Learn how to model and solve complex problems with computers.**

- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.
Goals for this Course

Learn how to model and solve complex problems with computers.

To that end:

- Explore common abstractions for representing problems.
- Harness recursion and understand how to think about problems recursively.
- Quantitatively analyze different approaches for solving problems.
What makes an algorithm “fast” or “slow”? 
Travel Time: 13 + 15 + 17 + 14 + 11 + 9 + 12 = 91
Travel Time: $10 + 17 + 7 + 14 + 13 + 4 + 7 = 72$
In an $n \times n$ grid, there are at least $4^n / n$ possible paths from one corner to another.
In an $n \times n$ grid, there are at least $\frac{4^n}{n}$ possible paths from one corner to another.

If $n = 50$, it would take the lifetime of the universe to list off all possible paths.
This approach is called **Dijkstra's Algorithm**.

Google Maps uses a slightly modified version of this algorithm. For an $n \times n$ grid, it requires (roughly speaking) $n \log n$ operations to find the shortest path.
This approach is called **Dijkstra's Algorithm**.

Google Maps uses a slightly modified version of this algorithm.
This approach is called **Dijkstra's Algorithm**.

Google Maps uses a slightly modified version of this algorithm.

For an $n \times n$ grid, it requires some multiple of $n^2 \log n$ operations to find the shortest path.
Goals for this Course

- **Learn how to model and solve complex problems with computers.**

- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.
Secondary Goal

- Get better at writing “good” code
  - What makes code “good”?
Example: Naming

```c
int numUnits(bool isGrad, bool wantsFewerUnits) {
    if (!isGrad) return 5;
    if (!wantsFewerUnits) return 5;
    if (reallyBusy()) {
        return 3;
    } else {
        return 4;
    }
}
```
Example: Naming

```c
int NU(bool isGrad, bool wantsFewerUnits) {
    if (!isGrad) return 5;
    if (!wantsFewerUnits) return 5;
    if (reallyBusy()) {
        return 3;
    } else {
        return 4;
    }
}
```
Example: Naming

```c
int NU(bool IG, bool WFU) {
    if (!IG) return 5;
    if (!WFU) return 5;
    if (reallyBusy()) {
        return 3;
    } else {
        return 4;
    }
}
```
Secondary Goal

- Get better at writing “good” code
- What makes code “good”?
Secondary Goal

- **Get better at writing “good” code**
  - What makes code “good”?
- One possible definition: code that's easy to understand, use and build upon
  - e.g. How hard would it be for someone to start working with this code?
Secondary Goal

- **Get better at writing “good” code**
  - What makes code “good”? 
  - One possible definition: code that's easy to understand, use and build upon
    - e.g. How hard would it be for someone to start working with this code? 
- Get better at this with practice, examples from class/section/course reader, advice from section leaders
One more detail...
What is C++?

- Programming language developed in 1983 by Bjarne Stroustrup.
- Widely used for general programming when performance is important.
- Supports a variety of programming styles.
C++ and CS106B

- The focus of CS106B is developing a set of problem solving skills.
- Learning C++ is not the focus of CS106B.
- We teach you just enough C++ in order to cover the topics in the course.
  - C++ just happens to be a useful language to cover these topics.
/* File: hello-world.cpp
 *
 * A canonical Hello, world! program
 * in C++.
 */

#include <iostream>
using namespace std;

int main() {
    cout << "Hello, world!" << endl;
}

#include ~ import
cout ~ println()
/* File: retain-evens.cpp

* A program to filter out odd numbers from a list.
*/
#include <iostream>
#include "vector.h"
using namespace std;

Vector<int> retainEvens(Vector<int> values) {
    Vector<int> result;
    for (int i = 0; i < values.size(); i++) {
        if (values[i] % 2 == 0)
            result += values[i];
    }
    return result;
}

int main() {
    Vector<int> values;
    values += 1, 2, 3, 4, 5;

    Vector<int> processed = retainEvens(values);

    for (int i = 0; i < processed.size(); i++) {
        cout << processed[i] << endl;
    }
}
C++

- Takeaway Point: Learning a new programming language is not like learning a new spoken language.
  - Most languages have similar features
CS106L

- Not offered over the Summer :(
- Optional, one-unit companion course to CS106B.
- In-depth treatment of C++'s libraries and language features.
- Excellent complement to the material from CS106B; highly recommended!
Having a Good Time in CS106B

• Start assignments early.
• Work during LAIR hours so you can ask a section leader if you have any questions.
• Go to section.
• Learn to use the debugger.
• Ask questions in lecture and section!
Next Time

• Welcome to C++!
  • Defining functions.
  • Reference parameters.
  • Introduction to recursion.