April 15, 2020
Road map

What’s it all about?

Course Topics

Policies & grades

Academic honesty

Help & advice
What’s it all about?
What CS 211 is all about (1/2)

From the course abstract:
What CS 211 is all about (1/2)

From the course abstract:

- *CS 211 teaches foundational software design skills at a small-to-medium scale.*
What CS 211 is all about (1/2)

From the course abstract:

- *CS 211 teaches foundational software design skills at a small-to-medium scale.* We will grow from writing single functions to writing interacting systems of several components.
What CS 211 is all about (1/2)

From the course abstract:

- *CS 211 teaches foundational software design skills at a small-to-medium scale.* We will grow from writing single functions to writing interacting systems of several components.
- *We aim to provide a bridge from the student-oriented HtDP languages*
What CS 211 is all about (1/2)

From the course abstract:

- **CS 211 teaches foundational software design skills at a small-to-medium scale.** We will grow from writing single functions to writing interacting systems of several components.
- **We aim to provide a bridge from the student-oriented HtDP languages (that is, CS 111)**
What CS 211 is all about (1/2)

From the course abstract:

- **CS 211 teaches foundational software design skills at a small-to-medium scale.** We will grow from writing single functions to writing interacting systems of several components.

- **We aim to provide a bridge from the student-oriented HtDP languages (that is, CS 111) to real, industry-standard languages and tools.**
What CS 211 is all about (1/2)

From the course abstract:

- **CS 211 teaches foundational software design skills at a small-to-medium scale.** We will grow from writing single functions to writing interacting systems of several components.

- **We aim to provide a bridge from the student-oriented HtDP languages (that is, CS 111) to real, industry-standard languages and tools.** Like C11, the UNIX shell, Make, C++14, and CLion.
What CS 211 is all about (1/2)

From the course abstract:

- **CS 211 teaches foundational software design skills at a small-to-medium scale.** We will grow from writing single functions to writing interacting systems of several components.

- **We aim to provide a bridge from the student-oriented HtDP languages (that is, CS 111) to real, industry-standard languages and tools.** Like C11, the UNIX shell, Make, C++14, and CLion.

- In the first half...
What CS 211 is all about (2/2)

From the course abstract:

- *In the first half, you’ll learn the basics of imperative programming and manual memory management using the C programming language.*
What CS 211 is all about (2/2)

From the course abstract:

- *In the first half, you’ll learn the basics of imperative programming and manual memory management using the C programming language.* This will help you form connections between the high-level programming concepts you learned in CS 111 and the low-level machine concepts you will learn in CS 213.

- *In the second half, we’ll transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas.*
What CS 211 is all about (2/2)

From the course abstract:

- *In the first half, you’ll learn the basics of imperative programming and manual memory management using the C programming language.* This will help you form connections between the high-level programming concepts you learned in CS 111 and the low-level machine concepts you will learn in CS 213.

- *In the second half, we’ll transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas.* We’ll learn how to define our own, new types that act like the built-in ones.
What CS 211 is all about (2/2)

From the course abstract:

- *In the first half, you’ll learn the basics of imperative programming and manual memory management using the C programming language.* This will help you form connections between the high-level programming concepts you learned in CS 111 and the low-level machine concepts you will learn in CS 213.

- *In the second half, we’ll transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas.* We’ll learn how to define our own, new types that act like the built-in ones.

- Topics include...
Course Topics
Topics

- Language mechanisms
- Design techniques
- Engineering practices
Topics

• Language mechanisms
  ▶ New syntax for functional programming

• Design techniques

• Engineering practices
Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions

- Design techniques

- Engineering practices
Topics

• Language mechanisms
  ▶ New syntax for functional programming: expressions, values, conditionals, variables, functions
  ▶ Imperative programming
    ▶ Statements: sequencing, iteration
    ▶ Mutation: objects, assignment

• Design techniques

• Engineering practices
Topics

• Language mechanisms
  ▶ New syntax for functional programming: expressions, values, conditionals, variables, functions
  ▶ Imperative programming
    ▶ Statements: sequencing, iteration
    ▶ Mutation: objects, assignment
  ▶ Memory allocation on the stack and the heap

• Design techniques

• Engineering practices
Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers

- Design techniques

- Engineering practices
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**

- **Engineering practices**
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types

- **Engineering practices**
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing

- **Engineering practices**
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAI

- **Engineering practices**
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization

- **Engineering practices**
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAIi: Resource Acquisition Is Initialization

- **Engineering practices**
  - Testing
Topics

• Language mechanisms
  ▶ New syntax for functional programming: expressions, values, conditionals, variables, functions
  ▶ Imperative programming
    ▶ Statements: sequencing, iteration
    ▶ Mutation: objects, assignment
  ▶ Memory allocation on the stack and the heap
  ▶ Representing information with structs, arrays, pointers
  ▶ Static types, type erasure, generics

• Design techniques
  ▶ Data abstraction: defining our own types
  ▶ Memory management via ownership and borrowing
  ▶ RAII: Resource Acquisition Is Initialization

• Engineering practices
  ▶ Testing: for gaining confidence in our software
Topics

• Language mechanisms
  ▶ New syntax for functional programming: expressions, values, conditionals, variables, functions
  ▶ Imperative programming
    ▶ Statements: sequencing, iteration
    ▶ Mutation: objects, assignment
  ▶ Memory allocation on the stack and the heap
  ▶ Representing information with structs, arrays, pointers
  ▶ Static types, type erasure, generics

• Design techniques
  ▶ Data abstraction: defining our own types
  ▶ Memory management via ownership and borrowing
  ▶ RAII: Resource Acquisition Is Initialization

• Engineering practices
  ▶ Testing: for gaining confidence in our software
  ▶ Debugging
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization

- **Engineering practices**
  - Testing: for gaining confidence in our software
  - Debugging: to see what’s happening in memory
Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization

- Engineering practices
  - Testing: for gaining confidence in our software
  - Debugging: to see what’s happening in memory
  - The Unix shell
Topics

- **Language mechanisms**
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics

- **Design techniques**
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization

- **Engineering practices**
  - Testing: for gaining confidence in our software
  - Debugging: to see what’s happening in memory
  - The Unix shell: a compositional user interface
Policies & grades
Grade composition

<table>
<thead>
<tr>
<th>what</th>
<th>%</th>
<th>when</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming homeworks</td>
<td>75*</td>
<td>Tuesdays</td>
<td>6</td>
</tr>
</tbody>
</table>

* Divided equally.
# Grade composition

<table>
<thead>
<tr>
<th>what</th>
<th>%</th>
<th>when</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming homeworks</td>
<td>75*</td>
<td>Tuesdays</td>
<td>6</td>
</tr>
<tr>
<td>final project proposal</td>
<td>5</td>
<td>We 5/13 – We 5/20</td>
<td>1</td>
</tr>
<tr>
<td>final project delivery</td>
<td>20</td>
<td>We 6/3</td>
<td>1</td>
</tr>
</tbody>
</table>

* Divided equally.
Grade composition

<table>
<thead>
<tr>
<th>what</th>
<th>%</th>
<th>when</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming homeworks</td>
<td>75*</td>
<td>Tuesdays</td>
<td>6</td>
</tr>
<tr>
<td>final project proposal</td>
<td>5</td>
<td>We 5/13 – We 5/20</td>
<td>1</td>
</tr>
<tr>
<td>final project delivery</td>
<td>20</td>
<td>We 6/3</td>
<td>1</td>
</tr>
</tbody>
</table>

* Divided equally.
Grade composition

<table>
<thead>
<tr>
<th>what</th>
<th>%</th>
<th>when</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming homeworks</td>
<td>75*</td>
<td>Tuesdays</td>
<td>6</td>
</tr>
<tr>
<td>final project proposal</td>
<td>5</td>
<td>We 5/13 – We 5/20</td>
<td>1</td>
</tr>
<tr>
<td>final project delivery</td>
<td>20</td>
<td>We 6/3</td>
<td>1</td>
</tr>
</tbody>
</table>

* Divided equally.
## Grade composition

<table>
<thead>
<tr>
<th>What</th>
<th>%</th>
<th>When</th>
<th>#</th>
<th>Drop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming homeworks</td>
<td>75*</td>
<td>Tuesdays</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Final project proposal</td>
<td>5</td>
<td>We 5/13 – We 5/20</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Final project delivery</td>
<td>20</td>
<td>We 6/3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*Divided equally.*
Homework policies

- Some will be done on your own
Homework policies

- Some will be done on your own
- Most will be pair-programmed with a *registered* partner
Homework policies

- Some will be done on your own
- Most will be pair-programmed with a registered partner
- Don’t be late
Homework policies

- Some will be done on your own
- Most will be pair-programmed with a *registered* partner
- Don’t be late
- You’ll need to do a self evaluation for each
Homework policies

- Some will be done on your own
- Most will be pair-programmed with a *registered* partner
- Don’t be late
- You’ll need to do a self evaluation for each
- No cheating...
Academic honesty
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outsider resource, such as Stack Overflow

- Please don't do these things, because:
  - If you don't write code, you won't learn; try to embrace the struggle!
  - All cheating will be reported to the relevant dean for investigation

If unsure about your particular situation, ask the instructor or other course staff.
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outsider resource, such as Stack Overflow

Please don’t do these things, because:
  - If you don’t write code, you won’t learn; try to embrace the struggle!
  - All cheating will be reported to the relevant dean for investigation

If unsure about your particular situation, ask the instructor or other course staff
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow

Please don’t do these things, because:
- If you don’t write code, you won’t learn; try to embrace the struggle!
- All cheating will be reported to the relevant dean for investigation
- If unsure about your particular situation, ask the instructor or other course staff
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow

- **Please don’t do these things**, because:
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow

- **Please don’t do these things**, because:
  - If you don’t write code, you won’t learn; try to embrace the struggle!
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow

- **Please don’t do these things**, because:
  - If you don’t write code, you won’t learn; try to embrace the struggle!
  - All cheating will be reported to the relevant dean for investigation
Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow

- **Please don’t do these things**, because:
  - If you don’t write code, you won’t learn; try to embrace the struggle!
  - All cheating will be reported to the relevant dean for investigation

- If unsure about your particular situation, ask the instructor or other course staff
Help & advice
Getting help

- **Synch.** Your course staff has office hours:
  
  Instructor: Jesse Tov (TuTh 4–6 PM CDT)
Getting help

- **Synch.** Your course staff has office hours:
  
  Instructor: Jesse Tov (TuTh 4–6 PM CDT)
  Grad TA: Mohammad Kavousi

The office hours schedule is a Google Calendar embedded on the course webpage: https://users.cs.northwestern.edu/~jesse/course/cs211/
Getting help

- **Synch.** Your course staff has office hours:
  
  Instructor: Jesse Tov (TuTh 4–6 PM CDT)
  Grad TA: Mohammad Kavousi

  The office hours schedule is a Google Calendar embedded on the course webpage: [https://users.cs.northwestern.edu/~jesse/course/cs211/](https://users.cs.northwestern.edu/~jesse/course/cs211/)

- **Asynch.** Ask questions on Campuswire: [https://campuswire.com/c/G123C6150](https://campuswire.com/c/G123C6150)
Getting help

- **Synch.** Your course staff has office hours:
  - Instructor: Jesse Tov (TuTh 4–6 PM CDT)
  - Grad TA: Mohammad Kavousi

The office hours schedule is a Google Calendar embedded on the course web page:

https://users.cs.northwestern.edu/~jesse/course/cs211/
Getting help

- **Synch.** Your course staff has office hours:
  - Instructor: Jesse Tov (TuTh 4–6 PM CDT)
  - Grad TA: Mohammad Kavousi

  The office hours schedule is a Google Calendar embedded on the course web page:

  https://users.cs.northwestern.edu/~jesse/course/cs211/

- **Asynch.** Ask questions on Campuswire:

  https://campuswire.com/c/G123C6150
Advice

• If you’re considering dropping, please talk to me first.
Advice

- If you’re considering dropping, please talk to me first.
- The only prerequisite is CS 111, so if you succeeded there then you absolutely do belong here.
Advice

• If you’re considering dropping, please talk to me first.
• The only prereq is CS 111, so if you succeeded there then you absolutely do belong here.
• If you find the course difficult, that’s because it is difficult.
Advice

- If you’re considering dropping, please talk to me first.
- The only prereq is CS 111, so if you succeeded there then you absolutely do belong here.
- If you find the course difficult, that’s because it is difficult.
- Be kind to each other.
I try not to make fun of people for admitting they don't know things. Because for each thing "everyone knows" by the time they're adults, every day there are, on average, 10,000 people in the US hearing about it for the first time.

Fraction who have heard of it at birth = 0%
Fraction who have heard of it by 30 ≈ 100%
US birth rate ≈ 4,000,000/year
Number hearing about it for the first time ≈ 10,000/day

If I make fun of people, I train them not to tell me when they have those moments. And I miss out on the fun.

"Diet Coke and Mentos thing"? What's that?

Oh man! Come on, we're going to the grocery store.

Why?

You're one of today's lucky 10,000.
Relative homework difficulties

<table>
<thead>
<tr>
<th>HW</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

(On a scale from 1 to 10)
Relative homework difficulties

(On a scale from 1 to 10)

<table>
<thead>
<tr>
<th>HW</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Relative homework difficulties

<table>
<thead>
<tr>
<th>HW</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

(On a scale from 1 to 10)
Relative homework difficulties

<table>
<thead>
<tr>
<th>HW</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

(On a scale from 1 to 10)
Relative homework difficulties

(On a scale from 1 to 10)
Relative homework difficulties

(On a scale from 1 to 10)
Relative homework difficulties

<table>
<thead>
<tr>
<th>HW</th>
<th>Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>FP</td>
<td>10ish*</td>
</tr>
</tbody>
</table>

(On a scale from 1 to 10)

* But really it’s up to you