Introduction

CS 211

Winter 2020

Road map

- What’s it all about?
- Topics
- Policies & grades
- Academic honesty
- Help & advice

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.
- We begin by learning…

What CS 211 is all about (2/2)

From the course abstract:

- *We begin by learning 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:

- *We begin by learning 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.
- *Then we 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:

- *We begin by learning 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.
- *Then we 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:

- *We begin by learning 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.
- *Then we 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…

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
  - RAII
- 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

Grade composition

what	**%**	**when**	**#**
programming homeworks	50*	Thursdays	6

* Divided equally.

Grade composition

what	**%**	**when**	**#**
programming homeworks	50*	Thursdays	6
final project proposal	5	Fr 2/21 – Th 2/27	1
two-week final project	15	Th 3/12	1

* Divided equally.

Grade composition

what	**%**	**when**	**#**
programming homeworks	50*	Thursdays	6
final project proposal	5	Fr 2/21 – Th 2/27	1
two-week final project	15	Th 3/12	1
in-class midterm exams	30*	Tu 2/4 & Tu 3/10	2

* Divided equally.

Grade composition

what	**%**	**when**	**#**
programming homeworks	50*	Thursdays	6
final project proposal	5	Fr 2/21 – Th 2/27	1
two-week final project	15	Th 3/12	1
in-class midterm exams	30*	Tu 2/4 & Tu 3/10	2
lab section attendance	0†	weekly	8

* Divided equally.

† May be used for close calls or to tweak weights in your favor.

Grade composition

what	%	when	#	drop
programming homeworks	50*	Thursdays	6	1
final project proposal	5	Fr 2/21 – Th 2/27	1	0
two-week final project	15	Th 3/12	1	0
in-class midterm exams	30*	Tu 2/4 & Tu 3/10	2	0
lab section attendance	0†	weekly	8	2

* Divided equally.

† May be used for close calls or to tweak weights in your favor.

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
- Late code will not be accepted

Homework policies

- Some will be done on your own
- Most will be pair-programmed with a *registered* partner
- Late code will not be accepted
- 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
- Late code will not be accepted
- You’ll need to do a self evaluation for each
- No cheating…

Academic honesty

In CS 211, we take cheating very seriously.

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Receive help of any kind on an exam (except from authorized course staff)

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Receive help of any kind on an exam (except from authorized course staff)
  - Give help of any kind on an exam

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
    - Receive help of any kind on an exam (except from authorized course staff)
    - Give help of any kind on an exam
    - Share (give or receive) homework code with anyone who is not your official, registered partner

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Receive help of any kind on an exam (except from authorized course staff)
  - Give help of any kind on an exam
  - Share (give or receive) homework code with anyone who is not your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Receive help of any kind on an exam (except from authorized course staff)
  - Give help of any kind on an exam
  - Share (give or receive) homework code with anyone who is not 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:
  - Receive help of any kind on an exam (except from authorized course staff)
  - Give help of any kind on an exam
  - Share (give or receive) homework code with anyone who is not 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:
  - Receive help of any kind on an exam (except from authorized course staff)
  - Give help of any kind on an exam
  - Share (give or receive) homework code with anyone who is not 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:
  - Receive help of any kind on an exam (except from authorized course staff)
  - Give help of any kind on an exam
  - Share (give or receive) homework code with anyone who is not 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

Getting help

- **In person.** Your course staff has office hours:

Instructor:	Jesse Tov (TuTh 3:30–4:30)

Getting help

- **In person.** Your course staff has office hours:

Instructor:	Jesse Tov (TuTh 3:30–4:30)
Grad TA:	Mohammad Kavousi

Getting help

- **In person.** Your course staff has office hours:

Instructor:	Jesse Tov (TuTh 3:30–4:30)
Grad TA:	Mohammad Kavousi
Peer TAs:	Ann Pigott, Brando Socarras, David Jin, Elise Lee, Margot Sobota, Max Chapin, Naythen Farr, Priya Kini

Getting help

- **In person.** Your course staff has office hours:

Instructor:	Jesse Tov (TuTh 3:30–4:30)
Grad TA:	Mohammad Kavousi
Peer TAs:	Ann Pigott, Brando Socarras, David Jin, Elise Lee, Margot Sobota, Max Chapin, Naythen Farr, Priya Kini

  The office hours schedule will be linked from the course web page:

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

Getting help

- **In person.** Your course staff has office hours:

Instructor:	Jesse Tov (TuTh 3:30–4:30)
Grad TA:	Mohammad Kavousi
Peer TAs:	Ann Pigott, Brando Socarras, David Jin, Elise Lee, Margot Sobota, Max Chapin, Naythen Farr, Priya Kini

  The office hours schedule will be linked from the course web page:

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

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

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

Advice

- If you’re considering dropping, come talk to me first.

Advice

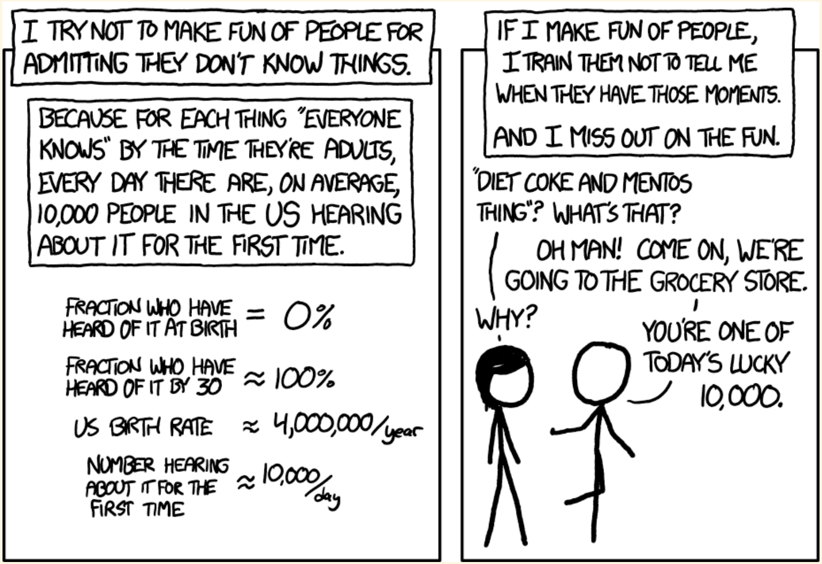
- If you're considering dropping, come talk to me first.
- The only prereq is CS 111, so if you succeeded there then you do belong here.

Advice

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

Advice

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

HW	**Difficulty**
1	2

Relative homework difficulties

HW	**Difficulty**
1	2
2	4

Relative homework difficulties

HW	**Difficulty**
1	2
2	4
3	7

Relative homework difficulties

HW	Difficulty
1	2
2	4
3	7
4	11

Relative homework difficulties

HW	Difficulty
1	2
2	4
3	7
4	11
5	4

Relative homework difficulties

HW	**Difficulty**
1	2
2	4
3	7
4	11
5	4
6	6

Relative homework difficulties

HW	Difficulty
1	2
2	4
3	7
4	11
5	4
6	6
FP	8ish

(On a scale from 1 to 10)

Prexamination!

Suppose each function is called with an arbitrary integer value. Circle *all possible* outcomes:

- T The function returns `true`
- F The function returns `false`
- A The program terminates abnormally (a crash!)

Prexamination!

Suppose each function is called with an arbitrary integer value.
Circle *all possible* outcomes:

- T The function returns `true`
- F The function returns `false`
- A The program terminates abnormally (a crash!)

```
bool g(int z)
{
    return false;
}
```

T F A

Prexamination!

Suppose each function is called with an arbitrary integer value.
Circle *all possible* outcomes:

- T The function returns **t**rue
- F The function returns **f**alse
- A The program terminates **a**bnormally (a crash!)

```
bool g(int z)
{
    return false;
}
```

T [✓ F] A

Prexamination!

Suppose each function is called with an arbitrary integer value. Circle *all possible* outcomes:

- T The function returns **t**rue
- F The function returns **f**alse
- A The program terminates **a**bnormally (a crash!)

```
bool h(int z)
{
    int y = z / 0;
    return false;
}
```

T F A

Prexamination!

Suppose each function is called with an arbitrary integer value. Circle *all possible* outcomes:

- T The function returns **t**rue
- F The function returns **f**alse
- A The program terminates **a**bnormally (a crash!)

```
bool h(int z)
{
    int y = z / 0;
    return false;
}
```

T **F** **A**