Introduction

EECS 211

Winter 2019

Road map

- What's it all about?
- Topics
- Policies
- Academic honesty
- How to get help

What EECS 211 is all about (1/2)

From the course abstract:

What EECS 211 is all about (1/2)

From the course abstract:

- *EECS 211 teaches foundational software design skills at a small-to-medium scale.*

What EECS 211 is all about (1/2)

From the course abstract:

- *EECS 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 EECS 211 is all about (1/2)

From the course abstract:

- *EECS 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 *curriculum*

What EECS 211 is all about (1/2)

From the course abstract:

- *EECS 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 *curriculum* (that is, EECS 111)

What EECS 211 is all about (1/2)

From the course abstract:

- *EECS 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 *curriculum* (that is, EECS 111) *to real, industry-standard languages and tools.*

What EECS 211 is all about (1/2)

From the course abstract:

- *EECS 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 *curriculum* (that is, EECS 111) *to real, industry-standard languages and tools.* Like C11, C++14, the UNIX shell, Make, and CLion.

What EECS 211 is all about (1/2)

From the course abstract:

- *EECS 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 *curriculum* (that is, EECS 111) *to real, industry-standard languages and tools.* Like C11, C++14, the UNIX shell, Make, and CLion.
- We begin by learning…

What EECS 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 EECS 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 EECS 111 and the low-level machine concepts you will learn in EECS 213.
- *Then we transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas.*

What EECS 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 EECS 111 and the low-level machine concepts you will learn in EECS 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 EECS 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 EECS 111 and the low-level machine concepts you will learn in EECS 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 basics

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works
- Structuring data

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works
- Structuring data: structs and vectors

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works
- Structuring data: structs and vectors
- The stack and the heap

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works
- Structuring data: structs and vectors
- The stack and the heap: how data is laid out and managed in memory

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works
- Structuring data: structs and vectors
- The stack and the heap: how data is laid out and managed in memory
- Data abstraction

Topics

- Language basics: expressions, statements, variables, types, assignment, control structures, functions
- Testing: how we know software works
- Structuring data: structs and vectors
- The stack and the heap: how data is laid out and managed in memory
- Data abstraction: using classes to define our own types

Policies

- There will be a homework assignment due every Thursday

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted
  - Best six of first seven worth 50% of your grade

- Two exams

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted
  - Best six of first seven worth 50% of your grade
  - Last two (final project) worth 20% of your grade
- Two exams

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted
  - Best six of first seven worth 50% of your grade
  - Last two (final project) worth 20% of your grade
- Two exams
  - Tuesday, February 5

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted
  - Best six of first seven worth 50% of your grade
  - Last two (final project) worth 20% of your grade
- Two exams
  - Tuesday, February 5
  - Tuesday, March 12

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted
  - Best six of first seven worth 50% of your grade
  - Last two (final project) worth 20% of your grade
- Two exams
  - Tuesday, February 5
  - Tuesday, March 12
  - Each worth 15% of your grade

Policies

- There will be a homework assignment due every Thursday
  - Some will be done on your own
  - Most will be pair-programmed with an assigned partner
  - Late work will not be accepted
  - Best six of first seven worth 50% of your grade
  - Last two (final project) worth 20% of your grade
- Two exams
  - Tuesday, February 5
  - Tuesday, March 12
  - Each worth 15% of your grade
- Mapping of point totals to letter grades is at instructor's discretion

Academic honesty

In EECS 211, we take cheating very seriously.

Academic honesty

In EECS 211, we take cheating very seriously.

- Cheating is when you:

Academic honesty

In EECS 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 EECS 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 EECS 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 partner

Academic honesty

In EECS 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 partner
  - Obtain code from an outside resource, such as Stack Overflow

Academic honesty

In EECS 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 partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things**

Academic honesty

In EECS 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 partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things**
  - If you don't write code, you won't learn; struggle is good

Academic honesty

In EECS 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 partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things**
  - If you don't write code, you won't learn; struggle is good
  - All cheating will be reported to the relevant dean for investigation

Academic honesty

In EECS 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 partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things**
  - If you don't write code, you won't learn; struggle is good
  - 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

Getting help

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

  Instructor: Jesse Tov
  Head TAs: German Espinosa, Samuel Hill

Getting help

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

Instructor:	Jesse Tov
Head TAs:	German Espinosa, Samuel Hill
Peer TAs:	Alex Rhee, Corinne Burger, Elise Lee, Finley Lau, Jayden Soni, Jordan Zax, Kevin Qiu, Kieran Bondy, Mario Lizano, Matt Cheung, Michael Cuevas, Michael Ji, Paul Farcasanu, Sarah O'Brien

Getting help

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

Instructor:	Jesse Tov
Head TAs:	German Espinosa, Samuel Hill
Peer TAs:	Alex Rhee, Corinne Burger, Elise Lee, Finley Lau, Jayden Soni, Jordan Zax, Kevin Qiu, Kieran Bondy, Mario Lizano, Matt Cheung, Michael Cuevas, Michael Ji, Paul Farcasanu, Sarah O'Brien

  Times and locations and will be listed on the course web page:

  `http://users.eecs.northwestern.edu/~jesse/course/eecs211/`

Getting help

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

Instructor:	Jesse Tov
Head TAs:	German Espinosa, Samuel Hill
Peer TAs:	Alex Rhee, Corinne Burger, Elise Lee, Finley Lau, Jayden Soni, Jordan Zax, Kevin Qiu, Kieran Bondy, Mario Lizano, Matt Cheung, Michael Cuevas, Michael Ji, Paul Farcasanu, Sarah O'Brien

  Times and locations and will be listed on the course web page:

  `http://users.eecs.northwestern.edu/~jesse/course/eecs211/`

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

  `https://piazza.com/northwestern/winter2019/eecs211`

Pop quiz!

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

C The function cannot be run, because the compiler rejects it

T The function returns `true`

F The function returns `false`

A The function causes the program to terminate abnormally

Pop quiz!

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

C The function cannot be run, because the compiler rejects it

T The function returns `true`

F The function returns `false`

A The function causes the program to terminate abnormally

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

C T F A

Pop quiz!

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

C The function cannot be run, because the compiler rejects it

T The function returns `true`

F The function returns `false`

A The function causes the program to terminate abnormally

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

C T **F** A

Pop quiz!

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

- C The function cannot be run, because the compiler rejects it
- T The function returns `true`
- F The function returns `false`
- A The function causes the program to terminate abnormally

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

C T F A

Pop quiz!

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

- C The function cannot be run, because the compiler rejects it
- T The function returns `true`
- F The function returns `false`
- A The function causes the program to terminate abnormally

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

C **T F A**