CSE 390B, Spring 2022

Building Academic Success Through Bottom-Up Computing

Stress & Wellness & Inclusive Design

Stress & Wellness Discussion, Overview of Inclusive Design, Design Decisions in Computing, Project 8 Overview

Lecture Outline

- **Stress & Wellness Discussion**
  - **Becoming Intentional about Wellness**
- Overview of Inclusive Design
  - Design Bias, Universal Design, Affordance Types
- Design Decisions in Computing
  - Accessibility, Technological Bias
- Project 8 Overview
  - Number Literal Example, Micro Jack Specification Notes, Tips

Stress & Wellness Podcast

- Reading: Episode 1 of the Feminist Survival Project podcast
- Topics: stress response cycle & dealing with stress versus dealing with stressors
- Lots of information to process from the podcast—what were your thoughts?

Stress & Wellness Discussion

- How much time do you set aside to intentionally take action towards pursuing your wellness?
- How much have you reflected on what actions you do that do and don't contribute to your wellness?
- What might be an example of treating wellness as a state of mind vs. treating wellness as a state of action?

Stress & Wellness Discussion

- Reflecting on your life, identify some of the common stressors that come up for you both externally (e.g., traffic) and internally (e.g., self-criticism)
- How do you typically respond to stressors (physically, emotionally, and cognitively)?
- In what areas of your life can you focus on addressing your stress response and your stressors?

Lecture Outline

- Stress & Wellness Discussion
  - Becoming Intentional about Wellness
- **Overview of Inclusive Design**
  - **Design Bias, Universal Design, Affordance Types**
- Design Decisions in Computing
  - Accessibility, Technological Bias
- Project 8 Overview
  - Number Literal Example, Micro Jack Specification Notes, Tips

What is Design?

- The way something works, including how someone uses it
  - Almost always includes some element of interaction
- Design can have different definitions, goals, and interpretations in different contexts
  - It's also not always about the end-user of a product
  - For example, you might design a codebase easier to maintain
- Everything we create has design, but there is a range to how intentional the design of something is
  - Could be completely forgotten about
  - Could be focused on throughout the creation of something

Why Talk About Design?

- Design dictates the interactions between us and everything around us

- Those interactions have a range of consequences
  - Positive: when you go to a website and you are easily able to find all the information you need
  - Unideal but harmless: if a person can't easily drink from a certain cup
  - Harmful: if a person can't easily use emergency equipment

Why Talk About Design?

- Seemingly harmless interactions can have real impact on people, especially if repeated
  - E.g., unable to use any door will make you feel unwelcome
- How can we design to create more positive reactions for more people while mitigating negative interactions?
  - Tough question in a world with so many diverse people
- What accountability should there be for more harmful interactions caused by the design of something?
  - A nontrivial question with a muddy web of answers

An Aside: Bias

- Biases are the beliefs we have, often formed by our experiences
  - Can be **explicit**: We consciously have a belief about something, and it may intentionally impact us
  - Can be **implicit**: Unconscious or impact us unintentionally

- We all have bias, and it is not inherently "good" or "bad"
  - Both potentially beneficial and potentially harmful consequences

- Eliminating bias is not a realistic goal
  - Attempting to mitigate negative consequences that come from bias is more realistic

Designer's Bias

- People often think of the "typical user" as someone who is similar to them or those they are close to
  - An example of the influence of their biases
- Even if we try to think beyond what is familiar to us, it is unlikely we will remove bias from the design process
  - Opinions about what something "should" do are inherently biased
- Ideally, we would develop processes that mitigate the negative effects of biases as much as possible
  - Recall biases can be both known (explicit) and unknown (implicit)

Bias and Design

- Following slides include some ideas and frameworks people have come up with related to bias and design

- Not meant to be the most important ideas
  - Think of it more as a few reference points that you can learn more about beyond this lecture
  - Discussions about bias and design are very nuanced and constantly evolving

- None of these ideas and frameworks solves these issues
  - But they can be used to think about them and build better practices

Universal Design

- Big idea: design things that can be used by as many people as easily as possible

- Designing things that work well for a wide range of people includes those who might usually be excluded
  - For example: video captioning

- The process of including everyone leads us to better design

Inclusive Design

- Including as diverse a range of perspectives when designing something as possible
  - Similar to universal design, but you may offer different solutions for different types of people (rather than one solution for all)
  - Including a diverse perspective does not just mean having a diverse team of people
  - It means valuing a diversity of opinions and experiences

- If we prioritize diverse perspectives, especially those that have been typically excluded, it will lead to things that benefit more people

Affordance Theory

- Framework for thinking about things around us
- Things provide different affordances to people
  - A way of defining what the capabilities of something are
- Can group these affordances into different categories:
  - What affordances does someone think something provides them?
  - What affordances does something actually provide someone?

Affordance Types

- **Perceptible affordance**: something does what someone thinks it can
- **Hidden affordance**: something does what someone thinks it can't
- **False affordance**: something doesn't do what someone thinks it can
- **Correct rejection**: something doesn't do what someone thinks it can't

Discussion of Design Principles in Practice

- What are some observations of design you have observed in the real world?
- What are some experiences you have had with technology that has privileged or discriminated against you?
- How might you design these technologies differently to be more inclusive?

Lecture Outline

- Stress & Wellness Discussion
  - Becoming Intentional about Wellness
- Overview of Inclusive Design
  - Design Bias, Universal Design, Affordance Types
- **Design Decisions in Computing**
  - **Accessibility, Technological Bias**
- Project 8 Overview
  - Number Literal Example, Micro Jack Specification Notes, Tips

Design in Computing

- Design discussions are relevant to computing
  - Many were developed with design in mind
- Technology can be biased
  - Design is part of almost everything in computing
  - Our biases influence the design of things
- The computer science field generally lacks diversity, which can lead to many harmful designs

Design in Computing: Accessibility

- Many group so researchers focused on making technology more accessible for people
  - E.g., making web pages easily navigable for people who are blind
  - E.g., expanding internet access to remote populations

- Connection: elements of both universal design and inclusive design
  - Universal design: Designing products that work for as many people as possible
  - Inclusive design: Including more perspectives in the design process, and potentially developing specific solutions aimed at including different groups of people

Design in Computing: Technological Bias

- Research related to bias in AI/ML algorithms
  - E.g.,facial recognition technology not working as well on people of color (trained on primarily white datasets)
  - E.g., racial bias in crime prediction algorithms (reflects the bias of our criminal justice system)

- These results biases reflect biased design decisions throughout development
  - Picking datasets biased towards certain communities
  - Testing applications in biased environments
  - Bias in what is prioritized within an algorithm

Moving Towards Inclusive Design

- Design is often categorized as being separate from other parts of the development process
  - Design occurs in every stage of product development
- You can voice feedback and concerns in design
  - You are ultimately contributing to the design of it
  - What conversations already occur, then ask how we can do better
- Different vision of how to approach building technology
  - Slogan offered by Animikii: “Move slow and empower people”

Moving Towards Inclusive Design

- Brief overview of design that only scratches the surface

- Entire fields and majors related to design and computing
  - Human Computer Interaction (HCI)
  - User Experience (UX/UI)
  - Human Centered Design and Engineering (HCDE major at UW)

- Related courses:
  - CSE 340: Interactive computing
  - CSE 440: Intro to HCI
  - SOC 225: Data and society
  - HCDE department has related courses too

Five-minute Break!

- Feel free to stand up, stretch, use the restroom, drink some water, review your notes, or ask questions
- We'll be back at:
- Research shows mid-lecture breaks reduce the decline of attention in the middle of lecture (Olmsted, 1999)

Olmsted III, John. "The Mid-Lecture Break: When Less Is More." *Journal of Chemical Education* (1999). https://pubs.acs.org/doi/abs/10.1021/ed076p525.

Lecture Outline

- Stress & Wellness Discussion
  - Becoming Intentional about Wellness
- Overview of Inclusive Design
  - Design Bias, Universal Design, Affordance Types
- Design Decisions in Computing
  - Accessibility, Technological Bias
- **Project 8 Overview**
  - **Number Literal Example, Micro Jack Specification Notes, Tips**

Project 8 Overview

- You will be given starter code for a compiler that reads a micro version of Jack and spits out Hack

- The Scanner & Parser are working
  - Task A: read through comments to understand what's going on

- The Code Generation is buggy and half-finished
  - Task B: find the bugs by practicing deliberate debugging strategies (e.g., step through generated Hack code using CPUEmulator)
  - Task C: Complete the implementation of the compiler

Project 8: Micro Jack

- Stripped-down version of Jack language
  - More manageable but enough features to be interesting
- Available features:
  - Types: Int and Int[], Variable Assignment, If, While, +, -, ==, !=
- Missing features:
  - Functions, function calls, classes, objects, strings, for loops, array bounds checking, etc.

Any number of variable declarations

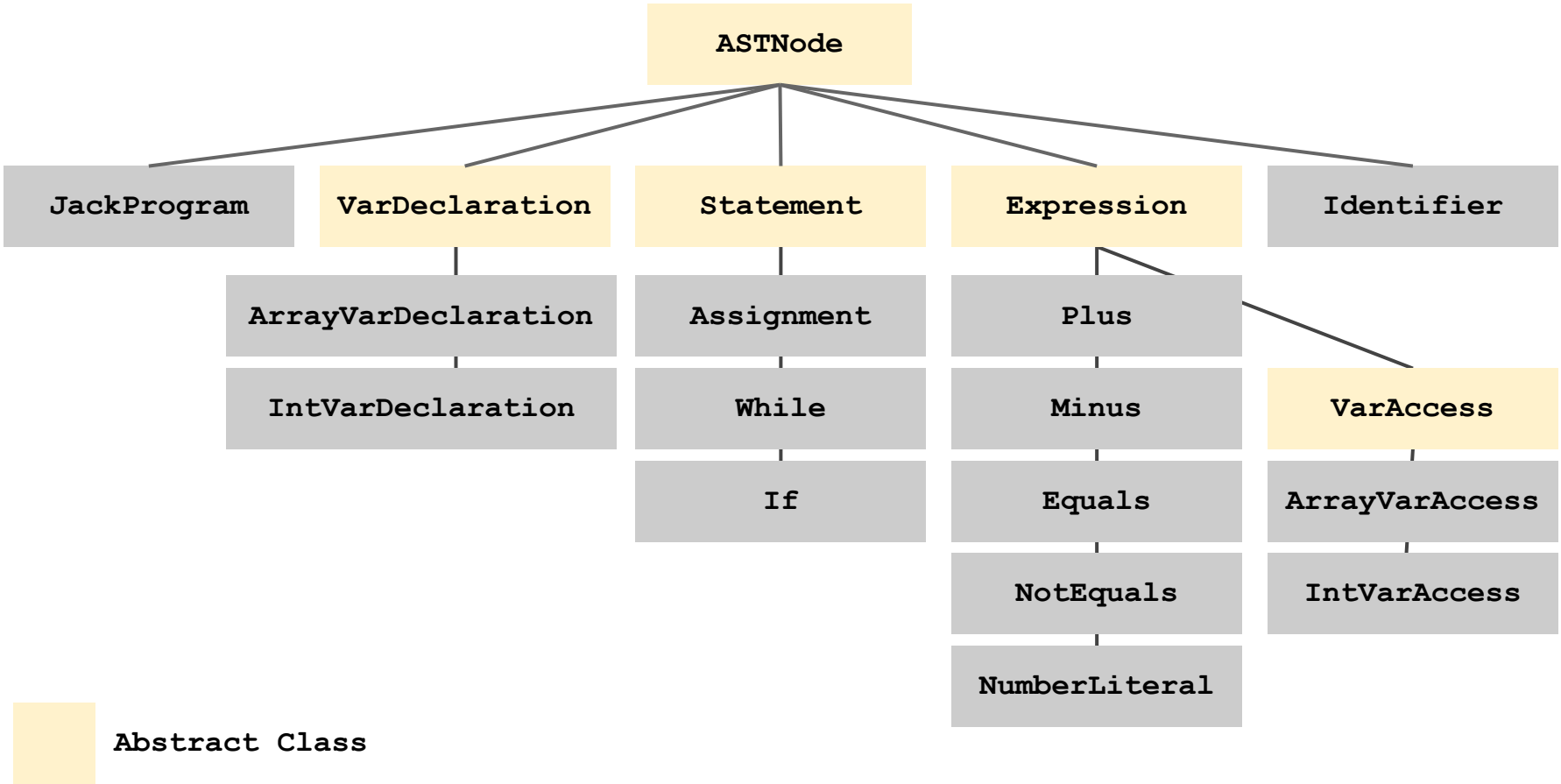
Basic.jack

```
var int a, b[1], c;
var int d[10], e;

let a = 1;
let b[0] = 1;
let n = 9;
while (n != 0) {
  let d[n] = a;
  let n = n - 1;
}
let screen[100] = d[0];
```

Then any number of statements

Project 8: The AST Nodes

- You are provided with all AST Node classes needed
  - All your code will be implemented within these classes



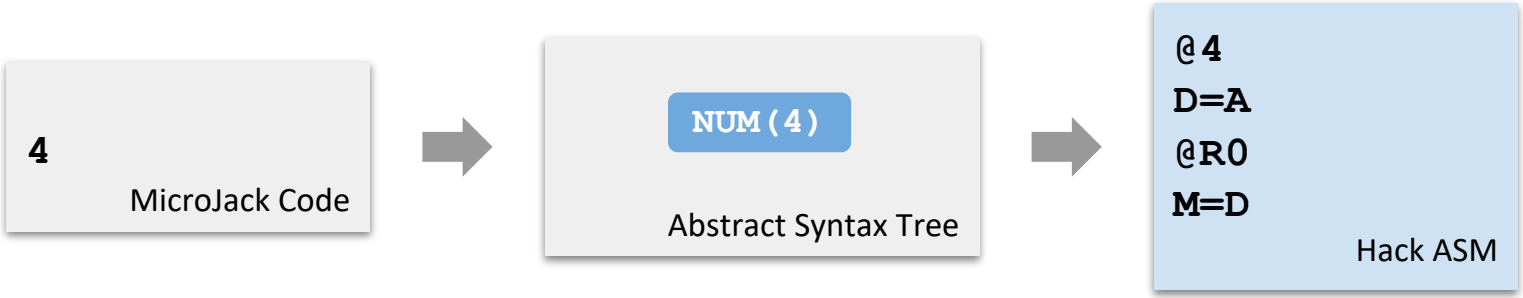
Project 8: Generating Code

- Each AST Node has a printASM method that should print out Hack instructions to System.out (and recursively call printASM on children)
  - You're provided with instr("@R0") and label("LOOP") convenience functions
  - Each can take a comment as a second argument—we highly recommend you take advantage of this

```
public class If extends Statement {
  public Expression condition;
  public List<Statement> statements;

  ...

@Override
public void printASM(symbolTable) {
  condition.printASM(symbolTable);
  instr("@R0", "Get cond result");
  instr("D=M");

  ...

}
```

Example: Number Literal (Step 1)

- Called a "literal" because it's a literal value embedded in the Micro Jack code
  - Generated Hack Assembly should simply put that value in R0



Example: Number Literal (Step 1)

```
public class NumberLiteral extends Expression {
    public int value;

    public NumberLiteral(String value) {
        this.value = Integer.parseInt(value);
    }

    @Override
    public void printASM() {
        comment("Start Number Literal");
        instr( ? );
        instr("D=A");
        instr("@R0");
        instr("M=D");
        comment("End Number Literal");
    }

    @Override
    public String toString() {
        return Integer.toString(value);
    }
}
```

4

```
// Start Number Literal
@4
D=A
@R0
M=D
// End Number Literal
```

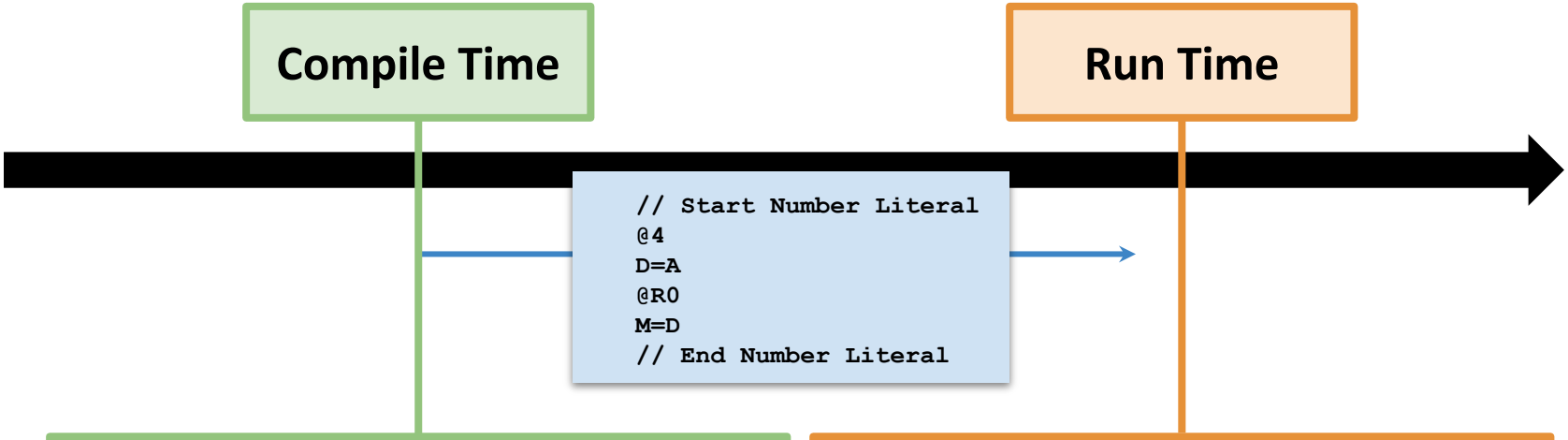
Example: Number Literal (Step 1)

```
public class NumberLiteral extends Expression {
    public int value;

    public NumberLiteral(String value) {
        this.value = Integer.parseInt(value);
    }

    @Override
    public void printASM() {
        comment("Start Number Literal");
        instr("@" + toString());
        instr("D=A");
        instr("@R0");
        instr("M=D");
        comment("End Number Literal");
    }

    @Override
    public String toString() {
        return Integer.toString(value);
    }
}
```

4

```
// Start Number Literal
@4
D=A
@R0
M=D
// End Number Literal
```

Example: Number Literal (Step 1)



- Java (the compiler) is running
- 4 is stored as value field inside an NumberLiteral ASTNode
- When executed, **prints code that stores it another way**!

```
public class NumberLiteral extends Expression {
    public int value;

    public NumberLiteral(String value) {
        this.value = Integer.parseInt(value);
    }

    @Override
    public void printASM() {
        comment("Start Number Literal");
        instr("@" + toString());
        instr("D=A");
        instr("@R0");
        instr("M=D");
        comment("End Number Literal");
    }

    @Override
    public String toString() {
        return Integer.toString(value);
    }
}
```

- Hack ASM (the output) is running
- 4 is stored as constant inside an assembly instruction
- When executed, loads 4 from instruction into A register

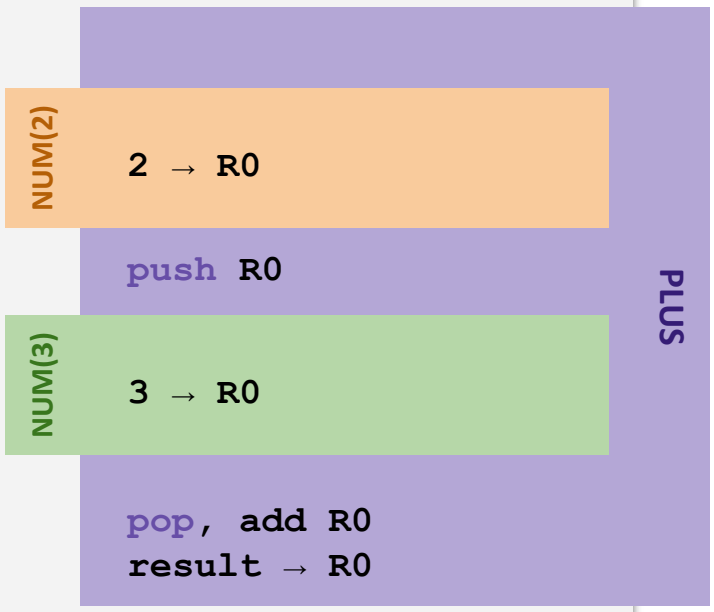
Example: Plus (Step 2)

```
public class Plus extends Expression {
    public Expression left;
    public Expression right;

    @Override
    public void printASM() {
        comment("Start Plus");

        left.printASM();

        instr("@R0");
        instr("D=M");

        right.printASM();

        push();

        instr("@R1");
        instr("A=M");

        instr("D=D+A", "perform the addition");

        ...
```

Example: Plus (Step 2)

1 Structural Bug: Map to abstract diagram for Plus:

```
public class Plus extends Expression {
    public Expression left;
    public Expression right;

    @Override
    public void printASM() {
        comment("Start Plus");

        left.printASM();

        instr("@R0");
        instr("D=M");

        right.printASM();

        push();

        instr("@R1");
        instr("A=M");

        instr("D=D+A", "perform the addition");

        ...
```



1 Detail Bug: Step through generated code, Check state at each step

Project 8 Buggy Compiler Overview

- Step 1: Read comments provided in the starter code
- Step 2: Implement NumberLiteral.java (~4 lines)
- Step 3: Debug Plus.java (2 bugs)
- Step 4: Implement Minus.java (~13 lines, similar to Plus.java)
- Step 5: Implement NotEquals.java (~21 lines, similar to Equals.java
- Step 6: Implement ArrayVarAccess.java (~3 lines)
- Step 7: Debug If.java (2 bugs)
- Step 8: Implement While.java (~14 lines)

Project 8: Micro Jack Specification Notes

- Can't write a negative integer literal
  - Instead, use subtraction from zero: 0 - 1
- All variable declarations must come before all regular statements
  - Why? Simplifies concept of a "defined" variable
- No defined operator precedence
  - If order matters for an operation, use parentheses

Project 8: Micro Jack Specification Notes

- Arrays are very simple
  - arr[index] just calculating an address: take address of arr variable and add index to it as an offset
  - No array bounds checking -- just lets you run off the end

- Booleans are just 0 (false) and non-zero (true)

Project 8: Debugging Tips

- Try walking through the general printASM code to understand why each line is there
  - Add comments to the assembly as you go! Much easier to understand resulting file
- Find the smallest example you can
  - Provided tests get progressively more complex, but you may want to write your own tiny test case to isolate
  - ASM gets long fast—we've added comments so you can isolate to the section you're working on
- "Play Computer": as you step through the code, write down the state you expect after each instruction, then advance and see if the CPUEmulator agrees

Additional Project 8 Tips

- When debugging assembly, a good first step is to try understanding the code and adding comments to the assembly as you go
  - Much easier to understand resulting file

- A printDebug method has been implemented for you on all AST Nodes
  - Use it to visualize exactly what the Parser is giving you, but also as a basis for printASM
  - Both need to do processing on the current node and strategically recurse on its children

Additional Project 8 Tips

- Pushing and popping from the stack can be intimidating, but formulaic
  - Understand it once, copy and paste afterward
  - push() and pop() are already implemented for you

- We provide only a few Micro Jack test files
  - We encourage you to write more of your own (think back to the debugging lecture)
  - Can use Sandbox.* to write more tests or create your own files

Project 8 Tools Demo

Project 8 Tools Practice

- Practice using the Project 8 tools! Try doing the following:
  - Run git pull to grab the Project 8 starter code
  - Navigate to the src/ directory: **`cd src/`**
  - Compile the Java source code of the compiler by running: **`javac $(find . -name "*.java")`**
  - Use your compiler to compile the Jack file for the OnlyVars program: **`java compiler/Compiler compile ../test/OnlyVars.jack`**
  - Load/run OnlyVars.tst in the CPUEmulator

- The above steps were taken from the "How to Run Tests" portion of the specification
  - Can refer to this when needed as you work through the project

Lecture 16 Wrap-up

- Home stretch on the horizon next week!
  - Metacognitive subjects: Stress & Wellness (continued), Overcoming Procrastination
  - Technical subjects: Operating Systems and Networking
  - Final Project Overview

- Project Reminders
  - **Project 7, Part I (Midterm Corrections) due tonight (5/18) at 11:59pm PDT (No late days)**
  - Project 7, Part II (Professor Meeting Report) due next Thursday (5/26) at 11:59pm PDT
  - Project 8 released, due Tuesday (5/31) at 11:59pm PDT