CSC 447 - Concepts of Programming Languages

Course Overview

Instructor: Eric J. Fredericks

Autonomous and Intelligent Systems

- Want a computer to perform a task (transfer money, order goods)
- Interact with the physical world (drive a car, fly an airplane)
- Requires clear **instructions**: a programming language
  - move left, right, up, down
  - do one step and then another step
  - repeat steps

PL Syntax, Semantics, Pragmatics

- **Syntax:** structure of a programming language and rules for writing valid instructions (grammar)
- **Semantics:** meaning of valid instructions
- **Pragmatics:** how to use the language, best practices for writing code

PL Syntax and Semantics

Different Syntax, Same Semantics

Java

```java
final int x = y>0 ? 1 : -1;
```

Scala

```scala
val x = if y>0 then 1 else -1
```

Same Syntax, Different Semantics

Java

```java
for  (var i = 0; i < 10; i++) {
  var x = i;
}
// Here i and x are both undefined
```

JavaScript

```javascript
for (var i = 0; i < 10; i++) {
  var x = i;
}
// Here i is 10 and x is 9
```

PL Concepts by Example

Java

```java
class Intro {
  private static int isEven(int n) {
    return n % 2 == 0 ? 1 : 0;
  }

  public static int countEven(List<Integer> xs) {
    int result = 0;
    if (!xs.isEmpty()) {
      result =
        isEven(xs.getFirst()) +
        countEven(xs.subList(1, xs.size()));
    }
    return result;
  }

  public static void main(String[] args) {
    List<Integer> numbers =
      Arrays.stream(args).
      map(str -> Integer.parseInt(str)).
      toList();
    System.out.println("You entered " + countEven(numbers) +
      " even numbers");
  }
}
```

Concepts

- Statements and expressions
- Strict and nonstrict evaluation
- Collection processing
- Recursion
- L-Values
- Argument passing
- Parametric polymorphism

PL Concepts by Example

C

```c
int isEven(int n) { return n % 2 == 0 ? 1 : 0; }

int countEven(int argc, const int* const xs) {
  int result = 0;
  if (argc > 0) {
    result =
      isEven(xs[0]) +
      countEven(argc-1, &xs[1]);
  }
  return result;
}

int main(int argc, char **args) {
  int *vals = malloc((argc-1)*sizeof(int));
  for (int i = 1; i < argc; ++i) {
    char* end;
    vals[i-1] = strtol(args[i], &end, 10);
    // forgot to check whether all characters
    // are consumed (i.e., end == '\0')
  }
  int numEven = countEven(argc-1, vals);
  printf("You entered %d even numbers\r\n", numEven);
  // forgot to free the dynamically allocated memory
  return 0;
}
```

More Concepts

- Safety and undefined behavior
- Static and dynamic memory allocation
- Scope and lifetime

PL Concepts by Example

Scala

```scala
enum List:
  case Empty
  case Cons (head:Int, tail:List)

  def length (result:Int = 0): Int = this match
    case List.Empty      => result
    case List.Cons(a,as) => length(as, result+1)

object List:
  def create(elems: Int*) =
    elems.foldRight(List.Empty)((e, l) => List.Cons(e, l))
```

More Concepts

- Algebraic datatypes
- Functional programming
- Pattern matching
- Tail recursion
- Type inference
- Currying

PL Pragmatics and Large Language Models

```
1 Implement a Java program that counts
2 how many even even numbers are in
3 the command line inputs
```

```java
public static void main(String[] args) {
  int evenCount = 0;

  for (String arg : args) {
    try {
      int number = Integer.parseInt(arg);
      if (number % 2 == 0) {
          evenCount++;
      }
    } catch (NumberFormatException e) {
      System.out.println(
        arg + " is not a valid integer.");
    }
  }
  System.out.println(
    "Count of even numbers: " + evenCount);
}
```

```
1 Implement a Scala program that counts
2 how many even even numbers are in
3 the command line inputs
```

```scala
def main(args: Array[String]): Unit = {
  val evenCount = args.foldLeft(0) {
    (count, arg) =>
      try {
        val number = arg.toInt
        if number % 2 == 0 then count + 1
        else count
      } catch {
        case e: NumberFormatException =>
          println(
            s"$arg is not a valid integer.")
          count
      }
  }
  println(
    s"Count of even numbers: $evenCount")
}
```

What are you doing to convince yourself that the code is correct?

Why is it Important to Understand PL Well?

- Write high-quality code
- Assess the correctness of (auto-generated) code
- Assess the quality of (auto-generated) code
- Assess the performance of (auto-generated) code
- Generalize (auto-generated) code
- Instruct tools to refactor and improve code
- Read, maintain, and extend existing code

Programming Languages

- **1970:** assembly, FORTRAN, COBOL, Lisp
- **1980:** C, Pascal, BASIC, ML, Smalltalk
- **1990:** C++, Perl, Objective C, Erlang
- **2000:** Java, JavaScript, Python, Ruby, Lua
- **2005:** C#
- **2010:** Scala, F#, Clojure, Go
- **2015:** Rust, Swift, Kotlin, Elm, Elixir, TypeScript, PureScript
- **2020:** ReasonML, Crystal, Pony, Zig
- Many more!

Programming Languages

In just a couple of minutes, we explored different implementations of the same feature in different languages!

- Programming languages keep popping up
- You will have to keep learning new languages
- And keep up with changes to current languages
  - Java 8 and C++ 11 added Lambda expressions (nested, anonymous functions)
- 💡 Lots of common concepts!

Learning Objectives

Learn PLs more easily by recognizing concepts

- "it has conditionals, recursion, loops"
- "it has closures"
- "it has list comprehensions"
- "it has dynamic dispatch"

- Deeper understanding of PL concepts / paradigms
- Impact of PL on program development, modularity, correctness, runtime efficiency, etc.
- Alternative way of thinking to double-check generated code

Non-Goals

- ✗ Learn a PL similar to one you already know
- ✓ Instead, learn concepts to facilitate quick assimilation of new PLs as they appear
- ✗ Learn an IDE
- ✓ Instead, become familiar with a range of tools and build systems

Course Overview

❗ This is a challenging course

- deepen your understanding about programming while also learning a new language
- study guides and extra credit will help you succeed!
- take advantage of office hours!

❓ What are different ways of expressing computations?

- **Programming paradigms and styles:**
  - functional vs object-oriented
  - mutability vs immutability
  - iteration vs recursion
  - pattern matching vs visitor pattern
- **Concepts:** lexical and dynamic scope; stack layout; inheritance and dynamic dispatch; nested structures (functions or objects); dynamic vs static type checking; subtyping; parametric polymorphism

Course Approach

- Hands on: write many programs and experiments
- Use AI coding support responsibly: write contracts and tests, develop language extensions to help analyze auto-generated code
- Scala as main language:
  - carefully designed *multi-paradigm* language
  - textbook explains PL concepts in context
- Also bits of: C, C++, C#, Java, JavaScript, … chosen as exemplars of concepts

Discord

- On Discord
- Use appropriate language

Asking Questions

- For non-personal messages use Discord
  - "I think there is a mistake with grading of Question 3 of Homework 2." - **direct message/email to instructor**
  - "I cannot run program `foo`. I have tried running it from the command line on OS X. See below for a transcript of what I typed and the error message I received. Could you please help?" - **ask in Discord #general channel**

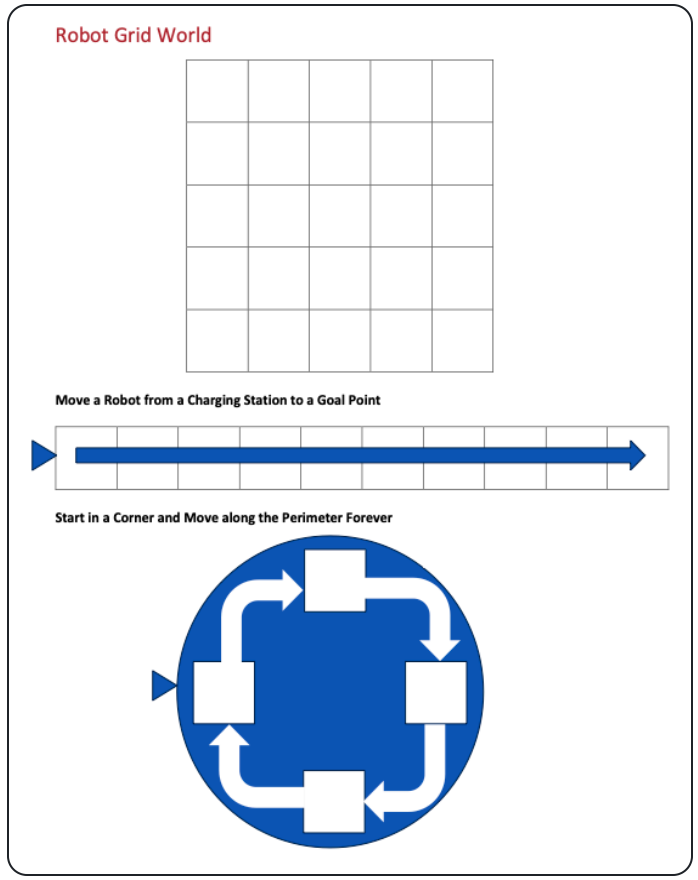
Asking Questions

- Include your actual (IRL) name in messages directly to the instructor
- Include enough context to answer your question:
  - "program `foo` doesn't work." - *starts discussion*
  - "program `foo` fails with output pasted below" - *good*
  - "program `foo` fails with input and output pasted below" - *better*
  - "program `foo` fails with input and output pasted below; using OS X; I have run extra commands to show current working directory, the version of `foo` in use, and other relevant information" - *best*

Course Syllabus

- **Review the Syllabus** linked from the course homepage
- Programming in Scala, First Edition is available for free online
- **Get Programming in Scala 5th edition**
- Earlier editions use a different version of Scala than class; you can use them but may need to look up new syntax

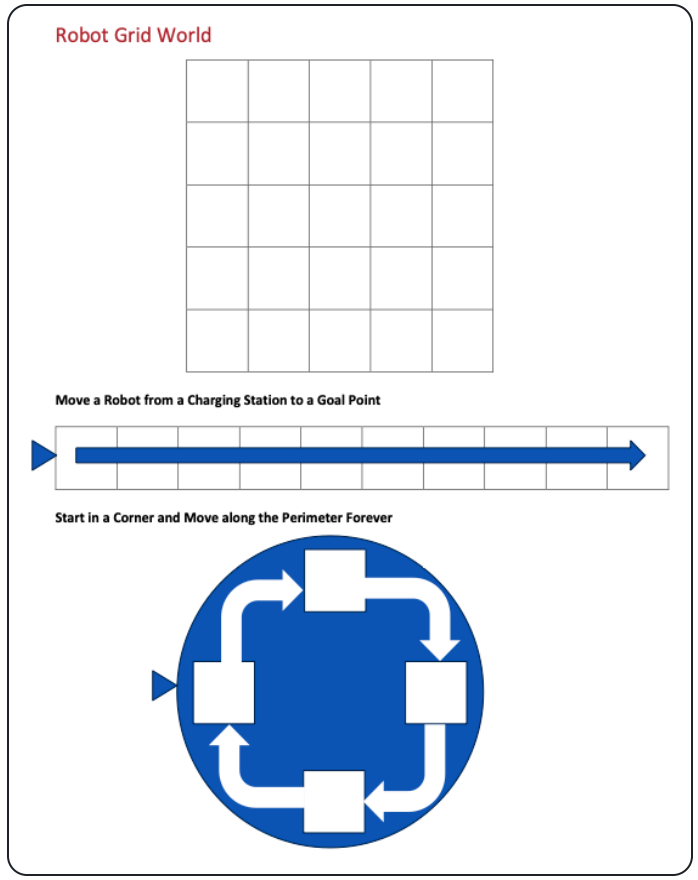
Program a Robot in a Grid World



Move Robot according to Instructions

- Mark the robot's initial position on the grid
- Mark a goal on the grid
- Create instructions to express how the robot moves
- Move the robot according to your instructions: fill in the *sequential* blue arrow instruction list

Program a Robot in a Grid World



Move Robot along the Perimeter

- Place the robot in a corner
- Use your instructions to express your program
  - The loop ↻ has 4 slots for instructions that repeat forever
  - Place your elementary instructions into the slots
- Move the robot according to your instructions

Computers that Program

- Teach a programming language to an LLM (ChatGPT)
  - Syntax (grammar) of the language

```
step ::= down n | up n | left n | right n | seq step then step (then step)* | repeat(step)
```

  - Semantics of the language
- Ask LLM to write a program

```
Use this language to move the robot in a 5 by 5 grid along the perimeter, starting in the top-left corner.
```

- Our programming language has
  - Parser to turn text into instructions
  - Interpreter to execute the instructions