CSE3302 Programming Languages
CSE5307 Programming Language Concepts

Kenny Q. Zhu
Dept. of Computer Science & Engineering
University of Texas at Arlington

Kenny Q. Zhu

Research Interests:

Artificial Intelligence

- Natural language understanding
- Natural language generation
- Knowledge representation/discovery

Programming Languages

- Domain specific languages
- Data Processing
- Concurrency

Recent Publications:

AAAI, IJCAI, ACL, EMNLP,…

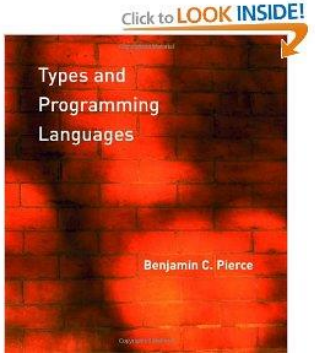
Degrees:	*National University of Singapore (NUS)*
Postdoc:	*Princeton University*
Experiences:	*Microsoft Redmond*
	Microsoft Research Asia
	Shanghai Jiao Tong University
	Joined *UT Arlington* in fall 2023

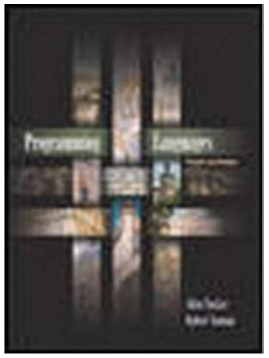
Administrative Info (I)

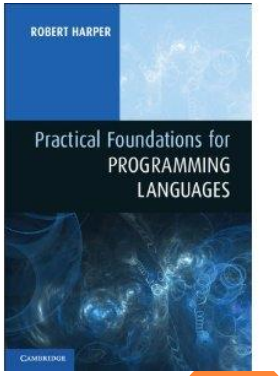
- Hybrid course (both undergrad & graduate)
- Lecturer:
  - Kenny Zhu, ERB-535, kenny.zhu@uta.edu
  - Office hours: Wed 4-5 PM, also by email appointments
- Teaching Assistant:
  - Essam Abdelghany, ERB-316, exa0039@mavs.uta.edu
  - Office hours: Thursday 10 AM-12 NOON
- Course Web Page (definitive source!): https://kenzhu2000.github.io/cse3302/
- Materials may be optionally uploaded to Canvas as well

Administrative Info (II)

- Format:
  - 1.5 hour lecture on Monday
  - 0.5 hour lecture and 1 hour tutorial discussion on Wednesday
  - Tutorials are led by TA
- Reference Texts:
  - **Types and Programming Languages** by Benjamin C. Pierce, The MIT Press.
  - **Programming Languages – Principles and Paradigms**, 2nd Edition, by Tucker & Noonan, McGraw Hill
  - **Practical Foundations for Programming Languages** by Robert Harper, Cambridge University Press
- Lecture materials on course web page







Administrative Info (III)

- 3-credit course (16 weeks)
- Modes of Assessment:
  - In-class quizzes: 10%
  - Tutorial discussion participation: 5%
  - Assignments: 30%
  - Programming Project: 25%
  - Final Exam: 30%
- Quizzes
  - Given out at random times
  - Usually on-screen **multiple-choice questions or short answer questions**
  - Bring piece of paper and a pen every time!
  - Submit answer after class (immediately) to TA or me
- Tutorials
  - Discuss assignment questions, issues in project, other Q&A
  - You will be asked to present your answers
  - Volunteer to win tutorial participation points

Administrative Info (IV)

- Assignments
  - Released (usually) every Wednesday)
  - Due date printed on assignment sheet
  - Submit solutions including code and data on Canvas
  - Late submission: -30% of full score for each additional day
  - Assignment solutions to be discussed at the tutorial in the following week (led by TA)

- Programming Project
  - Individual project
  - Implement an interpreter for a simple language called simPL
  - Be able to run test programs and produce correct evaluation results
  - Produce a report + code + results: due end of semester

Introduction

Why Do we Learn Programming Languages?

Two Misconceptions about This Course

- "This course about programming."

- "This is another compiler course."

Programming is about mastering the use of a language.
Compiler is about implementing a system that can parse a program in a high-level language into an intermediate form and then generate machine code. The focus is practical issues such as time and space complexity, code redundancy, and optimization.

What This Course is About

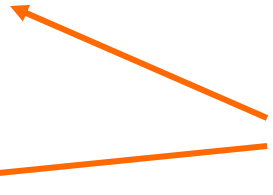
- *Theoretical aspects* of the design and implementation of all programming languages.
- The commonalities and differences between various *paradigms* and *languages*.
- So that you can:
  - Pick the right language for a project;
  - Design your own language (features);
  - Do programming language research.

Outline of Today's Lecture

- Principles
- Paradigms
- Special Topics
- A Brief History
- On Language Design
- Compilers and Virtual Machines
- Roadmap of This Course

The Factorial Program

- n! = 1 * 2 * … * n
- Or $n! = \prod_{i=1}^{n} i$



Two different mathematical languages

In computing, there are many more ways to do this …

The Factorial Program

C:

```
int factorial(int n) {
  int x = 1;
  while (n>1) {
            x = x * n;
            n = n -1;
  }
  return x;
}
```

Java:

```
class Factorial
{
   public static int fact(int n) {
      int c, fact = 1;
      if ( n < 0 )
       System.out.println("Wrong Input!");
      else {
         for ( c = 1 ; c <= n ; c++ )
            fact = fact*c;
         return fact;
      }
   }
}
```

The Factorial Program

Scheme:

```
(define (factorial n)
    (if (< n 1) 1
    (* n (factorial (- n 1)))
))
```

Prolog:

```
factorial(0, 1).
factorial(N, Result) :-
            N > 0, M is N - 1,
            factorial(M, SubRes),
Result is N * SubRes.
```

Principles

Programming languages have four properties:

- Syntax
- Names
- Types
- Semantics

For any language:

- Its designers must define these properties
- Its programmers must master these properties

Syntax

The *syntax* of a programming language is a precise description of all its grammatically correct programs.

When studying syntax, we ask questions like:

- What is the basic vocabulary?
- What is the grammar for the language?
- How are syntax errors detected?

Syntax

```
class Factorial
{
   public static int fact(int n) {
      int c, fact = 1;
      if ( n < 0 )
         System.out.println("Wrong Input!");
      else {
         for ( c = 1 ; c <= n ; c++ )
            fact = fact*c;
         return fact;
      }
   }
}
```

Vocabulary of Tokens:

- Literal (constant)
- Identifier
- Operator
- Separator (punctuation)
- **Reserved keyword**

Names

Various kinds of entities in a program have names:

variables, types, functions, parameters, classes, objects, …

An entity is bound to a name (identifier) within the context of:

- Scope (static/dynamic)
- Visibility (part of scope that is visible)
- Lifetime (dynamic and runtime)
- Type

Names

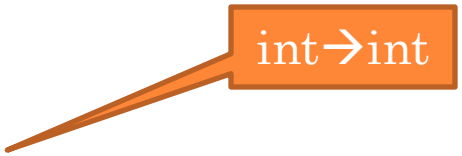
```
class Factorial
{
   public static int fact(int n) {
      int c, fact = 1;
      if ( n < 0 )
         System.out.println("Wrong Input!");
      else {
         for ( c = 1 ; c <= n ; c++ )
            fact = fact*c;
         return fact;
      }
   }
}
```

Types

A ***type*** is a collection of values and a collection of all *permissible* operations on those values.

- Simple types
  - numbers, characters, booleans, …
- Structured types
  - Strings, lists, trees, hash tables, …
- Function types
  - Simple operations like +, -, *, /
  - More complex/general function: int → int
- Generic types (polymorphism): α
- A language's *type system* can help:
  - Determine permissible (legal) operations
  - Detect type errors

TYPES




```
class Factorial
{
   public static int fact(int n) {
      int c, fact = 1;
      if ( n < 0 )
        System.out.println("Wrong Input!");
      else {
         for ( c = 1 ; c <= n ; c++ )
            fact = fact*c;
         return fact;
      }
   }
}
```