Lecture Outline

- Introduction: the rules
- Art of programming language design
- Programming language spectrum
- Why study programming languages?
- Compilation and interpretation

Read: Scott Chapter 1
Introduction

Homework is due at the beginning of lecture on the due date

- 6 late days in total
- 2 late days at most per homework

Extensions only with a formal excuse note from your Class Dean. Firm!

Fall 18 CSCI 4430, A. Milanova

Introduction

Quizzes

- 8 (or so) in-class quizzes
- Will cover material of previous week
- Work in groups is encouraged

- We will distribute quizzes at the beginning of lecture
- Keep and submit quiz at the end of class

Fall 18 CSCI 4430, A. Milanova

Introduction

Quiz or exam makeup will be arranged only after I have received a formal excuse note from your class dean. Firm!

Fall 18 CSCI 4430, A. Milanova

Introduction

In-class exercises

- I may give a simple problem on the material we have just covered at the end of class
- Work in groups and ask questions
- You can discuss your answers and receive feedback in office hours after class
- Participation in these exercises carries towards 2% extra credit

Fall 18 CSCI 4430, A. Milanova

Introduction

Graduate students taking 6430 as part of the PhD qualifying exam

- To earn an A, you must earn 96% or more
  AND
- Complete (satisfactorily!) a paper critique

Check syllabus

www.cs.rpi.edu/~milanova/csci4430/syllabus.htm for details

Fall 18 CSCI 4430, A. Milanova

Other Notes

Written homework: typically team homeworks

- Complete in teams of 2-3
- You must create a team in Submitty!
  - Identical submissions by individual students will be treated as cheating!

Programming homework: typically individual, largely autograded in Submitty

Fall 18 CSCI 4430, A. Milanova
Other Notes

- Submitty Discussion forum
  - Announcements – check regularly!
  - Ask all non-personal questions on the forum!
  - Check out prior messages before you post a question – the answer is probably already there!
- milanova@cs.rpi.edu (instructor) and mailing list csci4430@cs.lists.rpi.edu (instructor, TAs and mentors)
- Personal questions (extensions, grade disputes, etc.)
- Unsolicited debugging emails to instructor or mailing list will likely go unanswered!

Academic Integrity

- In short, do not copy and do not post your solutions/code on public forums or repos
- Excessive similarities between homework submissions will be considered cheating and handled accordingly
  - I trust you. Submitty has advanced plagiarism detection tools that course stuff runs regularly

How to Study

- Read textbook chapter in advance of lecture
  - I will assign reading at the end of each class
- Read lecture notes and textbook chapter (again) immediately after class
  - I will post lecture notes shortly after class
- Solve exercises in lecture notes
- Form study groups
- ASK QUESTIONS – in class, after class, on forum

Course Topics

- Programming language syntax: Scanning and parsing
- Programming language semantics: Attribute grammars
- Naming, binding and scoping
- Data abstraction and types
- Control abstraction and parameter passing
- Concurrency
- Logic-oriented language: Prolog
- Functional languages: Scheme and Haskell
- Imperative languages
  - An object-oriented language: Java
  - A dynamic language: Python
- Scala
Lecture Outline

- Introduction to the course
- The art of programming language design
- Programming language spectrum
- Why study programming languages?
- Compilation and interpretation

The Art of Language Design

- Why are there so many languages?
  - Different programming domains
  - Business applications
    - Process data, reports, transactions, increasingly distributed
  - Manufacturing/control systems
    - Reliability and non-stop operation
  - Scientific applications
    - Computationally intensive
  - Software is constantly put to new uses!
  - Personal preference…

The Programming Language Spectrum

- Imperative languages
  - Von Neumann languages: Fortran, C,…
  - Object-oriented languages: Java, C++, Smalltalk,…
  - Dynamic languages: Perl, Python, PHP,…
- Declarative languages
  - Functional languages: Scheme/Lisp, ML, Haskell
  - Logic languages: Prolog
  - There are other declarative languages: e.g., dataflow languages

Imperative languages

- Evolved from the von Neumann Architecture
- Variable
- Assignment Statement
- "von Neumann bottleneck": tube connecting CPU and memory
### The Programming Language Spectrum

- **Imperative languages**
  - Most widely popular programming style
  - FORTRAN, C, C++, C#, Java, Python, Visual BASIC, Perl, JavaScript, Ruby, etc.
  - Variable and assignment statement are central concepts
  - Program is a sequence of statements:
    
    \[
    j := i - j; \;
    k := j \times l; \;
    \]
  - Execution is a sequence of transitions on memory state

- **FORTRAN** was invented in mid-1950
- John Backus, the inventor of FORTRAN, wrote the following paper in 1979:
  “Can programming be liberated from the von Neumann style? A functional style and its algebra of programs”

- **Problems with imperative languages**
  - Difficult to understand programs
  - Difficult to reason about correctness of programs
  - Von Neumann Bottleneck

### Functional Programming

- Main alternative to imperative programming
  - Lisp/Scheme, ML/Ocaml, Haskell
  - Program consists of function definitions + evaluation expr
    
    \[
    (\text{fun3} (\text{fun2} (\text{fun1} \text{data}))) \;
    (\text{fun3} (\text{fun2} \text{data2})) \;
    (\text{fun3} \text{data3}) \;
    \text{data4} \;
    \]
  - Execution is a sequence of function applications (i.e., reductions)

- **Logic Programming**
  - Perform queries against knowledge base
  - Prolog, Datalog, SQL

### An Example: Inner Product

- **Inner product in FORTRAN:**
  
  1. \( C := 0; \)
  2. for \( I := 1 \) step 1 until \( N \) do
  3. \( C := C + a[I] \times b[I]; \)

- **Illustrates state-transition semantics**

- **Inner product in FP:**
  
  Def \( IP = (\text{Insert} +)^{0} (\text{ApplyToAll}^{*}) (\text{Transpose}) \)
  
  \( IP <<1,2,3>,<6,5,4>> \) is
  
  (Insert +) ((ApplyToAll *) (Transpose <<1,2,3>,<6,5,4>>))
  (Insert +) ((ApplyToAll *) <<1,6>,<2,5>,<3,4>>)
  (Insert +) <6,10,12>

- **Illustrates reduction (applicative) semantics**

### Why Study Programming Languages

- **Goal of the course:** learn to analyze programming languages
  - What are the questions we ask when facing a new programming language
  - Helps learn new languages, choose the right language for a problem, understand language features, design better languages!
Lecture Outline

- Introduction to the course
- The art of language design
- The programming language spectrum
- Why study programming languages
- Compilation and interpretation

Compilation and Interpretation

- Compilation
  - Compiler
    - A "high-level" program is translated into executable machine code
  - Interpretation
    - Interpreter
      - A program is translated and executed one statement at a time
  - Hybrid interpretation
    - Both a compiler and an interpreter
      - A program is "compiled" into intermediate code; intermediate code is "interpreted"

Compilation

- Scanner
  - Character stream
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
  - Token stream
  - Scanner (lexical analysis)
Interpretation

A language can be implemented using a compiler or using an interpreter
- One can build a compiler for Lisp and one can easily build an interpreter for C or Fortran
- However, language features (determined during language design) have significant impact on “compilability” and the decision “compiler vs. interpreter”

Compilation vs. Interpretation

Advantages of compilation?
- Faster execution

Advantages of interpretation?
- Greater flexibility
  - Sandboxing, source-level debugging, dynamic semantic (type) checks, other dynamic features are much easier

New languages are increasingly dynamic
- Hybrid implementations are common: compiler from source to intermediate form, then a smart interpreter (Virtual Machine) which does interpretation and JIT (Just-in-Time) compilation
- Tons of code written in static languages (C, C++, Fortran)
- Dynamic instrumentation tools (Valgrind, Pin, DynamoRio) serve as “interpreters” for compiled binaries
Next Class

- We will review regular expressions and context free grammars

- Read Chapter 2.1 and 2.2 from Scott's book