CS 314 Principles of Programming Languages

Topic: Introduction

Professor Louis Steinberg

Contacts

- **Prof. Louis Steinberg**
  - **lou @ cs.rutgers.edu**
  - **401 Hill**
- **TAs:**
  - **To be Announced**
- **Class web site: Sakai**

Recitation Starts Next Week

- **There is no recitation this week. Recitation will start January 26 or 29.**

No Laptops or Phones

- **In lecture, please put away all laptops, phones, tablets, etc.**
  - **Too much of a temptation**
- **If you have a disability and need a laptop to take notes, please see me.**

Text Book

- **Michael L. Scott, *Programming Language Pragmatics*, 3rd edition**
  - **recommended, not required**
- **Shriram Krishnamurthi, *Programming Languages: Application and interpretation*, 1st edition**
  - **available free at**
    http://cs.brown.edu/~sk/Publications/Books/ProgLangs/2007-04-26/plai-2007-04-26.pdf
  -

Piazza

- **We will use Piazza.com for online questions, answers, and discussions.**
- **Everything should be asked on Piazza, not email.**
- **Except**
  - **Please do not post source code from assignment or project solutions. Email these.**
  - **Please do not ask duplicate questions. Use search.**

Work

- **2 Midterms**
- **Final**
- **4 (?) projects**
- **Homework**
  - **pass/fail**
  - **“I choose not to do this homework.” => pass**

- **See Sakai Resources for exam dates, rules**

Tentative Weights

- **Midterm 1 20%**
- **Midterm 2 20%**
- **Final 35%**
- **4 (?) projects 15% total (but will also be tested on exams)**
- **Homework 10% total (but will also be tested on exams)**

Prerequisites

- **Prerequisites:**
  - **CS 205**
  - **CS 211**
- **I assume you already know**
  - **Java**
  - **The memory model of C (pointers)**
  - **Predicate calculus**

Topics

We will cover (tentatively):

- **Several metaphors for programming**
  - **Functional programming (Scheme)**
  - **Logic programming (Prolog)**
  - **Scripting (Python, shell??, javascript??)**
  - **(??) Parallel Programming (OpenMP)**
- **Several topics that apply to all languages, including**
  - **Grammars**
  - **Parameter passing modes**
  - **Types and type checking**

Main purpose

- **The main purpose of 314 is to teach you new ways to think about problems and programs**
- **The secondary purpose of 314 is to practice learning new languages so you can learn others when you need to**
- **The tertiary purpose of 314 is to introduce a couple of useful languages**

Anthropomorphism

- **Whenever a human term (e.g., ‘language’) is used about computers, ask:**
- **How analogous**

- **How differs**

Anthropomorphism

- **Whenever a human term (e.g., ‘language’) is used about computers, ask:**
- **How analogous**
  - **has a syntax: what is legal, semantics: what it means**
  - **express an algorithm**
- **How differs**
  - **syntax rigid, simple, unambiguous**
  - **meaning unambiguous**
  - **express only an algorithm**
  - **you don’t “say” a program, you design and construct it (but you don’t say an essay or a novel either)**

Does the Language Matter

- In some sense, any program in any major language can be translated into any other major language
- So does it really matter what language you use?

- Yes:
  - Language features <=> ways to think about problems
    - E.g., recursion
  - Language features => easier or harder to make mistakes
    - Eg case statements => forget a break

Desiderata for PL Design

- **Readable**
  - **comments, names, syntax, …**
- **Simple to learn**
  - **Orthogonal - small number of concepts combine regularly and systematically (without exceptions)**
- **Portable**
  - **language standardization**
- **Abstraction**
  - **control and data structures that hide detail**
- **Errors detectable early**
- **Compilable to efficient enough code**

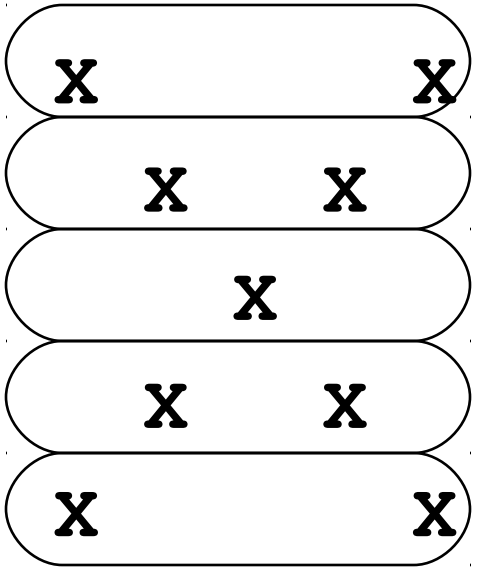
What is a Paradigm

- **A way of looking at a problem and seeing a program**
  - **What kind of parts do you look for?**
- **Problem: Print a "large X" of size n.**
  - **E.g., size 5 is**

```
X       X
  X   X
    X
  X   X
X       X
```

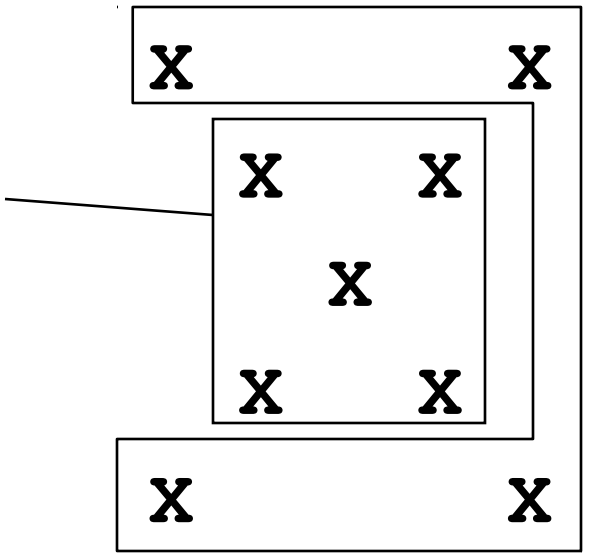
What is a Paradigm

- A way of looking at a problem and seeing a program
  - What kind of parts do you look for?
- E.g., iteration
  - line by line



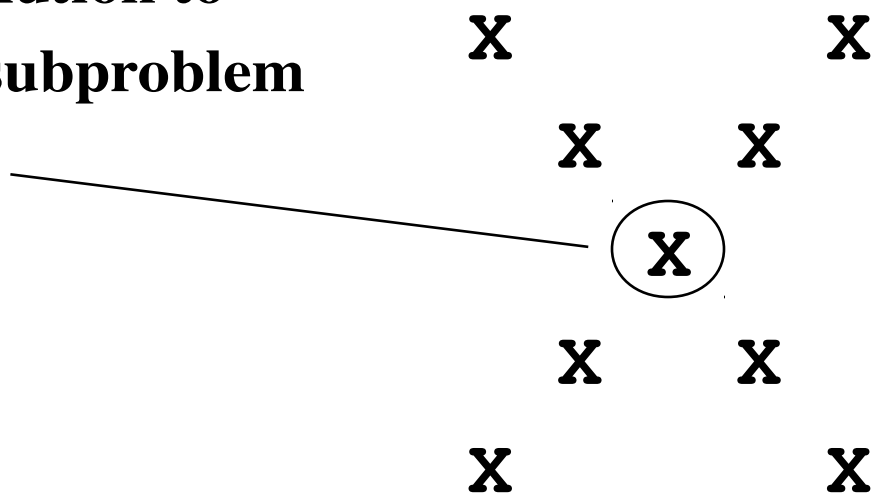
What is a Paradigm

- A way of looking at a problem and seeing a program
  - What kind of parts do you look for?
- E.g., recursion
  - Part of solution is solution to similar but smaller subproblem



What is a Paradigm

- A way of looking at a problem and seeing a program
  - What kind of parts do you look for?
- E.g., recursion
  - Part of solution is solution to similar but smaller subproblem
  - Also a terminal case



Imperative Paradigm

- **A program is: A sequence of state-changing actions**
- **Manipulate an abstract machine with:**
  - **Variables that name memory locations**
  - **Arithmetic and logical operations**
  - **Reference, evaluate, assign operations**
  - **Explicit control flow statements**
- **Fits the Von Neumann architecture closely**
- **Key operations:** ***Assignment, Call***
  - **also** ***Go To, Go To if 0***
  - **or** ***If, While***

Imperative Paradigm

Sum up twice each number from 1 to N.

Fortran

```
      SUMX2 = 0
      DO 11 K=1,N
      SUMX2 = SUMX2 + 2*K
11    CONTINUE
```

C

```
sumx2 = 0;
for (k = 1; k <= n; ++k)
  sumx2 += 2*k;
```

Pascal

```
sumx2 := 0;
for k := 1 to n do
  sumx2 := sumx2 + 2*k;
```

Functional Paradigm

- **A program is: Composition of functions on data**
- **Characteristics (in pure form):**
  - **Name values, not memory locations**
    - **bind rather than assign: a variable is a table entry not a memory location**
  - **Value binding through parameter passing**
  - **Recursion rather than iteration**
- **Key operations: *Function Application* and *Function Abstraction***
  - **Based on the Lambda Calculus**

Functional Paradigm

Scheme

```
(define (sumx2 n)
   (if (= n 0)
      0
      (+ (* n 2) (sumx2 (- n 1)))
   )
)

(sumx2 4) evaluates to 20
```

Logic Paradigm

- **A program is: Formal logical specification of problem**
- **Characteristics (in pure form):**
  - **Programs say *what* properties the solution must have, not *how* to find it**
  - **Solutions are obtained through a specialized form of *theorem-proving***
- **Key operations: *Unification* and *NonDeterministic Search***
  - **Based on First Order Predicate Logic**

Logic Paradigm

```
sumx2(0,0).
sumx2(N,S) :-
    N>0,
    NN is N - 1,
    sumx2(NN, SS),
    S is N * 2 + SS.
```

Prolog rules

```
?- sumx2(1,2).
yes
?- sumx2 (2,2).
no
?- sumx2(4,S).
S = 20
```

Queries and results

Object-Oriented Paradigm

- **A program is: Communication between abstract objects**
- **Characteristics:**
  - **"Objects" collect both the data and the operations**
  - **"Objects" provide *data abstraction***
  - **Methods can be either imperative or functional (or logical)**
- **Key operation: *Message Passing* or *Method Invocation***

Object-Oriented Paradigm

```
class IntNode{
    int data;
    IntNode next;
    public IntNode(int data,
                   IntNode next){
        this.data = data;
        this.next = next;
    }
}
```

```
public class IntLL {
    IntNode front;
    public IntLL( ) {
        front = null;
    }
    public void addToFront(int data) {
        front = new IntNode(data, front);
    }
}
```

Java

Why Learn More than One Programming Language?

- **Each language encourages thinking about a problem in a particular way.**
- **Each language provides (slightly) different expressiveness & efficiency.**

  ⇒ The language should match the problem.

Size does matter

- **Programs can be written by**
  - **one person**
  - **a small group (2 – 10)**

    ….
  - **a really large group**
    - **linux kernel: 10,000**
- **Programs can be used by**
  - **one person once**
  - **one person over time**
  - **a small group**
  - **many people**

Size Matters

- **Programs can be**
  - **a few lines**
  - **a few hundreds of lines (a few pages)**

    ….
  - **Millions of lines (e.g. linux kernel)**

Size Matters

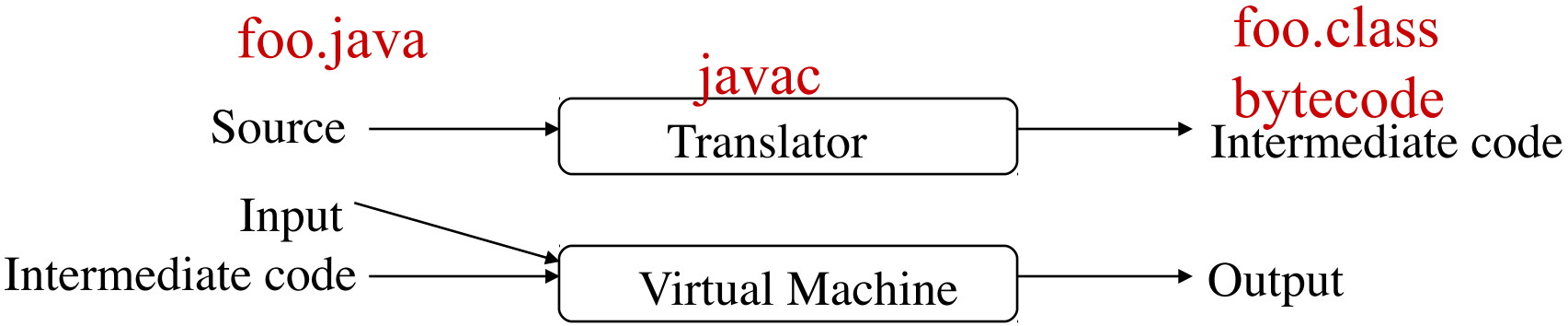
- As the scale goes up, complexity goes up
- Need new tools
  - debuggers
  - version control (like Git)
- Need new language features
  - e.g. block structured naming
  - e.g. multi-file programs

Translation

- **Computers don't understand high level languages**
- **So to get our program to run it must be translated (by a program!!)**
- **Many features of programming languages are to ease translation**

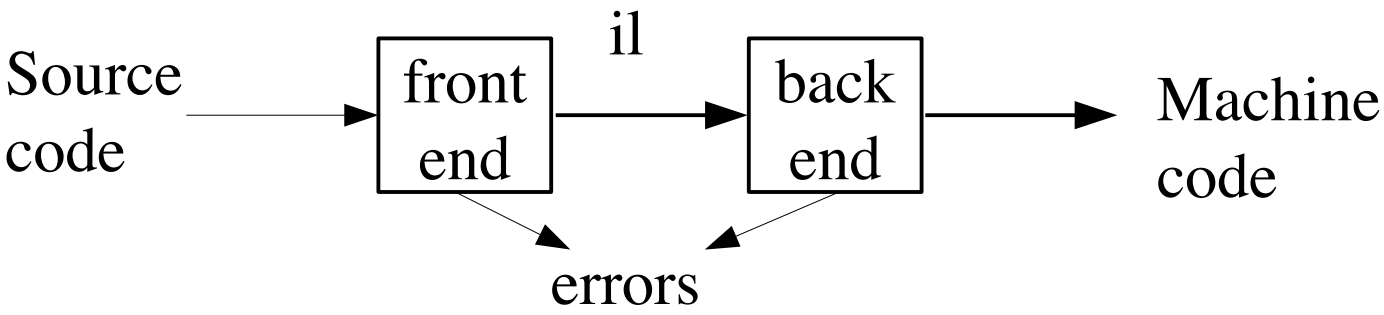
Translation

- ***Compilation*: Program is translated from a high-level language into a form that is executable on an actual machine**
- ***Interpretation*: Program is translated and executed one statement at a time by a "virtual machine"**
- **Some PL systems are a mixture of these two**
  - **E.g., Java**



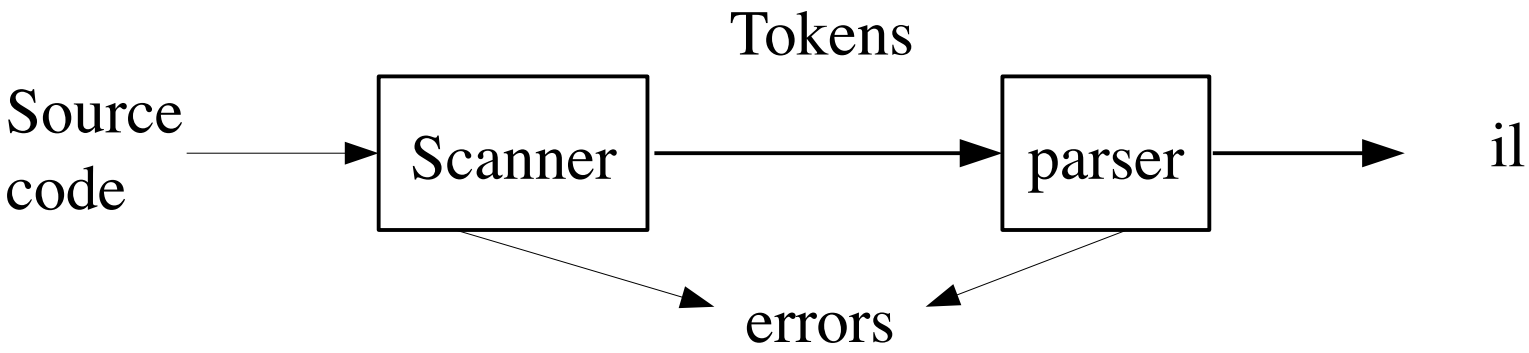
Traditional Two-Pass Compiler

- **Pass: read and write entire program**



- **il = intermediate language**
- **Simplifies retargeting to a new machine**
- **Can have multiple front ends for multiple languages**
- **More passes → slower compiling, faster running code**

Front end



Scanner

- **Maps characters → tokens**
  - **Tokens: basic unit of syntax**
  - **E.g., x = x + y; becomes**

    <id, x> <operator, assign> <id, x> <operator + > <id, y>
  - **Typical token types:**

    number, id, operator (e.g., +), keyword (e.g., do, else)

Parser

- **Parse: determine the grammatical structure of a sequence of tokens**
- **Grammar: set of rule like**

  assignment → variable '=' expression ';'
- **Analogy with grammars of human languages.**
  - **e.g., English: Sentence → Subject Verb Object**

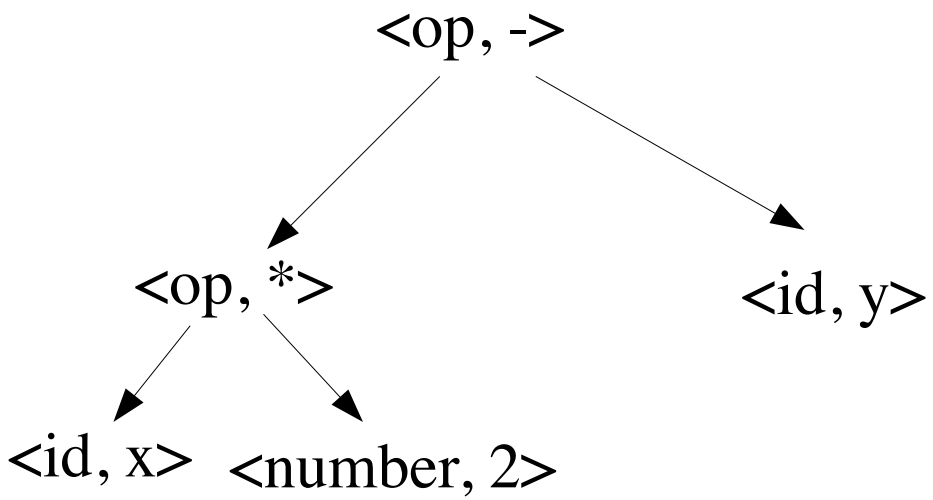
    The dog bit Bob

    Bob bit the dog ← different meaning

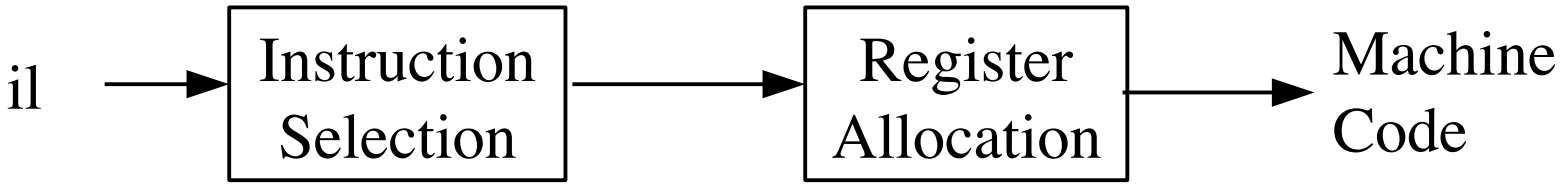
    Dog Bob the bit ← meaningless

Abstract Syntax Tree

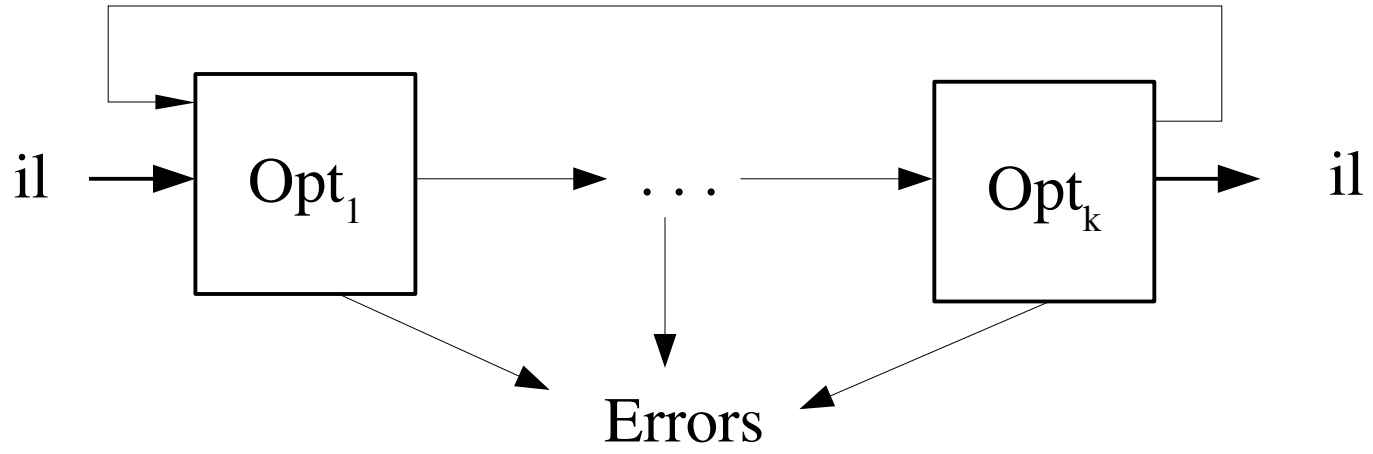
For: x * 2 - y



Back end


il
Instruction Selection
Register Allocation
Machine Code

Optimizers



- **Between front end and back end**
- **A set of passes, e.g.:**
  - **Discover & propagate constant values**
  - **reduction of operator strength (* → +)**
  - **common subexpression elimination**
  - **move computation out of loops**

So Many Languages

- **A diagram of 50 languages**
  - **See http://www.levenez.com/lang/history.html**
- **A list of 2500 languages**

  http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

To do

- **Read Scott, Chapter 1 (covers this lecture)**
- **Read Scott, Sections 2.1 and 2.2**
- **Sign up on Piazza (see our Sakai site)**
- **ßHomework 1 – see Sakai > Assignments**

Recitation Starts Next Week

- **There is no recitation this week. Recitation will start January 26 or 29.**