Lecture 1. Overview

January 26, 2021

"Science attempts to find logics and simplicity in nature; mathematics attempts to establish order and simplicity in human thoughts"

- Edward Teller

COM S 342: Principles of Programming Languages

Wei Le, Associate Professor in Computer Science

My goal: maximize your learning on the topic of programming languages

Syllabus: How I run the course

It is a foundational course in Computer Science and Software Engineering

- help you find jobs (functional programming, language and compiler design for domain-specific languages)
- make you a better programmer: select an appropriate language for the task, help write efficient code
- improve your ability to quickly learn a language in the future
  - Apple: Swift
  - Google: Go and Dart
  - Microsoft: F# and TypeScript
  - Mozilla: Rust
- provoke formal and deep thinking in computing

Outline

- What is a programming language?
- history and types of programming languages
- programming paradgims and styles
- designing programming languages

What is a programming language

Intuitively

- a language to write programs with; programs are setences and paragraphs written in the language

- **syntax**: defining "words" and rules for composing words
- **semantics**: defining the meaning of the words; in computation, the "meaning" is the value that the "words" compute
- how to design such a language: design decisions and tradeoffs

- how to specify such a language (syntax and semantics): for precise communication and automating the implementation
- how to design a software to automatically interpret/compile the programs written in this language to generate output

What is a programming language?

More formally

- a language that expresses computations
- a language in which developers write code/instructions for computers

Programming language vs. natural Language

- Both have certain structures, i.e., syntax and grammar.
- Natural language can be ambigious, but it is not a desired property for programming languages. (because we use software to translate programs to executable code while NLP is understood by human or machine intelligence)
- There is *naturalness* in programs (programmers introduce some natural language flavor when writing code, e.g., function names, variable names, certain grammar rules are used more often than others)

History of programming languages

1950s: FORTRAN, LISP, COBOL (NASA, ATMs, credit card)
1970s: PASCAL, C (Unix)
1980s: C++ (Firefox, Chrome, Adobe, IE)
1990s: Python, Java (Android)

10 top programming languages:

- 2021: JavaScript, Python, C/C++, Java, R, Kotlin (functional programming features)
- 2019: JavaScript, Python, Java, C/C++, PHP, Swift, C#, Ruby, Objective-C, SQL
- 2018: Python, C++, C, Java, C#, PHP, R, JavaScript, Go, Assembly

Types of programming languages

One classification:

- **general-purpose language**: express all computation
- **domain-specific language (DSL)** : support data types, relations, operations in domain
  - the Dot language for Graphviz – purpose: graph visualization, special concepts: nodes/edges
  - the HTML language for browsers – purpose: display web pages, special concepts: markup or typesetting related concepts
  - the SQL language for database – purpose: query database, special concepts: support query, join database

Another classification:

- **assembly language**
- **high level language**: programs in high-level languages are eventually translated to machine level via *Compilation*, *Interpretation* or *Hybrid*

Programming paradigms, programming styles

Ways of thinking about computation:

- Imperative: Fortran, Pascal, C (express computation using instructions)
- Object-oriented: Smalltalk, C++, Java (using objects)
- Functional: ML, Ocaml, Haskel, Scheme, Scala, Lisp, R ... (using functions)
- Logic: Prolog (using facts and relations)

'functional programming' (FP) is a programming style in which mathematical (partial) functions are used as the core programming abstraction. Functional languages make this programming style more natural.

Topics of this course: write code beyond C, Java and Python

Scheme (Racket) and Prolog (SWI-Prolog)
Example: Appending two lists

Append a List: Racket

Editor:

```
1   ; append fn takes two lists as param
2   (define append
3     (lambda (lst1 lst2)
4   ; if first lst is empty return second list
5       ( if (null? lst1) lst2
6   ; if second list is empty return first list
7         ( if (null? lst2) lst1
8           ; else build recursive logic to append two lists
9           ( cons (car lst1) (append (cdr lst1) lst2))
10        )
11      )
12    )
13  )
```

Console:

```
 > (append '() '(1 2 3))
=> (1 2 3)
 >
```

Append a List: SWI-Prolog

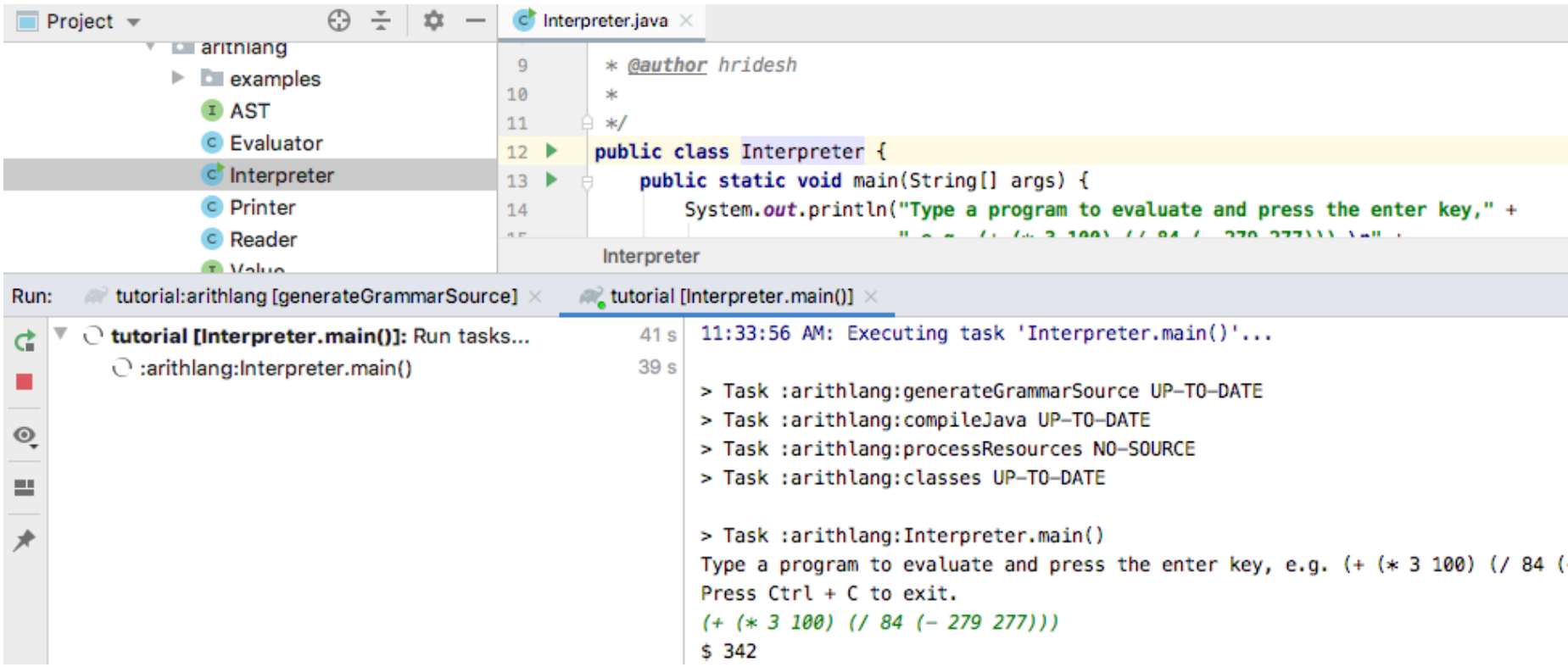
Editor:

```
1 % Demo 1: swi-prolog
2 % Write a program to append numbers from two input lists into a single output list.
3
4 % The append function involves 3 lists, the output list and two input lists.
5 % Starts with what is the initial condition.
6 append([], L, L).
7
8 % Next, the function specifies relationship between these 3 lists.
9 append([X|Xs], L, [X|Ys]):- append(Xs, L, Ys).
```

Console:

```
append([], [1,2], [1,2]).
true.
append([], [1,2], X).
X = [1, 2].
```

Topics of this course: implement your own programming languages (functional programming languages) using imperative and OO programming language (Java)



Imperative programming

- + Easier to learn, taught more often
- + Better development environments (IDE) and libraries
- + Typically faster
- - Side effect, hard to reason modularly
- - Hard to parallel?

Functional programming

- + side-effect free and easy to reason about: input and output completely describes the behavior of any function
- + less code
- - less efficient?
- - less support for IDE and libraries
- - hard to learn, not taught in school often

Logic programming

- Data as facts and relations
- Computations as logical inferences
- Control constructs: if-then-else and recursion
- High level description of computation

Reverse a list

Imperative Programming

```
void reverse(struct node** head_ref)
{
    struct node* prev    = NULL;
    struct node* current = *head_ref;
    struct node* next;
    while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
}
```

Functional Programming

```
(define (rev lst)
   (if (null? lst)
     lst
     (append (rev (cdr lst))
             (list (car lst))
     )
   )
)
```

Logic Programming

```
rev([], []).
rev([H|T, L) :-
   rev(T, T1),
   append(T1, [H], L).
```

Desiging a programming language: consider three parts

- **Computation**: to actually compute, e.g. primitive expressions, addition, subtraction, multiplication
- **Composition**: to put together computation, e.g., sequential (order), choice, or repeat
- **Abstraction**: to make programming scalable, e.g., function, name, that can be repeatedly used to refer to a complex piece of computation

Designing a programming language: a tool for communication

- Syntax for validity: is this sentence syntacitically valid? parser errors
- Semantics for understanding: what does this sentence mean? to computers: what is the value of this sentence?

How to specify a language

Lanuague users, language designers, language implementer may not be able to talk to each other directly. So written formal specification (without ambiguity) is needed:

1. English prose and examples in a careful, expository document (ambiguous, corner cases)
2. compiler/interpreter implementation
3. Formal, mathematical tools: grammar, semantics

Taste of formalism: mathematical aspects of programming languages

- context free grammars
- operational semantics
- lambda calculus: using functions to represent computation

Given
true: $\lambda x.(\lambda y.x)$
false: $\lambda x.(\lambda y.y)$
$\neg$: $\lambda x.((x\ \mathit{false})\mathit{true})$

Prove the following:

1. $\neg$ false = true
2. $\neg(\neg\text{true}) = \text{true}$

After lecture:

- HW0: get the framework code running to prepare future homework, will be taught in the recitation next week
  - download framework code and tutorials from canvas
  - install Java and IntelliJ
  - import the project
  - success point: you can compile and run projects on your machine
- Further reading for your interest: Ray Tracer Language Comparison