Language Design and Evolution

Programming Languages
William Killian
Millersville University
What is a Language?
language

/ˈlaNGgwij/

noun

1. the method of human communication, either spoken or written, consisting of the use of words in a structured and conventional

2. a system of symbols and rules for writing programs or algorithms
Linguistics Definition

the method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way

• Does “spoken” make sense for computers?

• Is “human communication” feasible with computer programs?
Computer Science Definition

*a system of tokens and rules for writing programs or algorithms*

- What tokens do we use?
- What rules do we have?
In the domain of language design, we will refer to each word as a unique *token*. Tokens can come in any order or be anything, but some tokens might have some special meaning. Usually alphanumeric, numeric, or symbolic.

What tokens might we have in Java? C++?
Keywords

• **Keywords** are a type of Token
• Usually reserved by the language
• Only be used in specific locations

Examples:

- `int` in Java
- `void` in C++
- `const` in Javascript
- `def` in Python
- `let` in OCaml

*Others?*
Symbols

- **Symbols** are another type of Token
- Usually combinations of punctuation characters
- Often used to indicate special operations

Examples:

- `==` in Java
- `|>` in OCaml
- `...` in Javascript
- `<<` in C++

Others?
Identifiers

• **Identifiers** refer to specific entities of our program
• Creating a new variable or function
• Accessing a data member
• Calling a function
• Using a library
Rules

• Within language design, we will refer to the order in which symbols can be structured as rules

• Two main types of rules:
  • Syntactic Rules
    The order of all symbols must be well-formed
  • Semantic Rules
    The meaning of all symbols must be well-formed

• We will talk about these in detail later in the class
The boy went to school.
Syntax Rules (in Java)

```
class      ::=  CLASS  ID  classArg*  LBRACE  defs*  RBRACE
classArg   ::=  (IMPLEMENTES|EXTENDS)  nameList
nameList   ::=  ID  [,  nameList]*
defs       ::=  [visibility]  [STATIC]  (varDef  SEMI|  funDef)
visibility ::=  (PUBLIC|PROTECTED|PRIVATE)
varDef     ::=  type  ID
funDef     ::=  type  ID  LPAREN  [paramList]  RPAREN  ...
paramList  ::=  varDef  [,  paramList]*
```
Syntax Rules (in Java)

```java
class := CLASS ID classArg* LBRACE defs* RBRACE
classArg := (IMPLEMENTS | EXTENDS) nameList
nameList := ID [, nameList]*
defs := [visibility] [STATIC] (varDef SEMI | funDef)
visibility := (PUBLIC | PROTECTED | PRIVATE)
varDef := type ID
funDef := type ID LPAREN [paramList] RPAREN ...
paramList := varDef [, paramList]*
type := (primitive | ID)
primitive := (INT | BOOLEAN | DOUBLE | SHORT | LONG | BYTE | FLOAT)
```

CAPS means terminal (symbol)
italicized means rule
* - zero or more
[optional]
(choice1|choice2)
Syntax Rules (in Java)

class := CLASS ID classArg* LBRACE defs* RBRACE
classArg := (IMPLEMENTS | EXTENDS) nameList
nameList := ID [, nameList]*
defs := [visibility] [STATIC] (varDef SEMI | funDef)
visibility := (PUBLIC | PROTECTED | PRIVATE)
varDef := type ID
funDef := type ID LPAREN [paramList] RPAREN ...
paramList := varDef [, paramList]*
type := (primitive | ID)
primitive := (INT | BOOLEAN | DOUBLE | SHORT | LONG | BYTE | FLOAT)

What are the tokens in the shown rules?
Syntax Rules (in Java)

```
class := CLASS ID classArg* LBRACE defs* RBRACE
classArg := (IMPLEMENTS | EXTENDS) nameList
nameList := ID [, nameList]*
defs := [visibility] [STATIC] (varDef SEMI | funDef)
visibility := (PUBLIC | PROTECTED | PRIVATE)
varDef := type ID
funDef := type ID LPAREN [paramList] RPAREN ...
paramList := varDef [, paramList]*
type := (primitive | ID)
primitive := (INT | BOOLEAN | DOUBLE | SHORT | LONG | BYTE | FLOAT)
```

Where are the rules?
Syntax Rules (in Java)

```plaintext
class := CLASS ID classArg* LBRACE defs* RBRACE
classArg := (IMPLEMENTS | EXTENDS) nameList
nameList := ID [, nameList]*
defs := [visibility] [STATIC] (varDef SEMI | funDef)
visibility := (PUBLIC | PROTECTED | PRIVATE)
varDef := type ID
funDef := type ID LPAREN [paramList] RPAREN ...
paramList := varDef [, paramList]*
type := (primitive | ID)
primitive := (INT | BOOLEAN | DOUBLE | SHORT | LONG | BYTE | FLOAT)
```

Exercise: Make a “Point” class with two public members of type double: x and y?
Rosetta Code Examples (Homework)

• What problems did you investigate?

• What languages did you see?

• What languages **DON’T** you want to see again?
Language Transformation

- Computers understand binary
  - Sequence of 0’s & 1’s
  - See: Computer Architecture
- Humans understand *languages*
  - See: Programming Languages

We need a *tool* to translate our language to binary
Compilers

• Input:
  • A program (sequence of instructions) written in a well-defined, predictable language

• Output:
  • A sequence of bits that a computer architecture can execute

• Task:
  • Compilers translate from an input language to an output language without loss of functionality
  • Mathematically correct
The First Programming Language

Discussion:

• What year do you think the first programming language came out?
• What did it look like?
• What did it run on?

Before 1930
1930 – 1940
1940 – 1950
After 1950
The First Programming Language

Ada Lovelace, 1843
The First Programming Language

Created

Plankalkül (1942-1945)

• Konrad Zuse

P1 max3 (V0[:8.0], V1[:8.0], V2[:8.0]) → R0[:8.0]
  max(V0[:8.0], V1[:8.0]) → Z1[:8.0]
  max(Z1[:8.0], V2[:8.0]) → R0[:8.0]
  END

P2 max (V0[:8.0], V1[:8.0]) → R0[:8.0]
  V0[:8.0] → Z1[:8.0]
  (Z1[:8.0] < V1[:8.0]) → V1[:8.0] → Z1[:8.0]
  Z1[:8.0] → R0[:8.0]
  END

function max3 (in a linear transcription) that calculates the maximum of three variables
The First Programming Language

Commercial

FORTRAN (1954)

• John Backus + IBM

C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - TAPE READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING

READ INPUT TAPE 5, 501, IA, IB, IC

501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE OR ZERO
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C MUST BE GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO

IF (IA) 777, 777, 701
701 IF (IB) 777, 777, 702
702 IF (IC) 777, 777, 703
703 IF (IA+IB-IC) 777, 777, 704
704 IF (IA+IC-IB) 777, 777, 705
705 IF (IB+IC-IA) 777, 777, 799
777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE

799 S = FLOATF (IA + IB + IC) / 2.0
799 AREA = SQRTF(S * (S - FLOATF(IA)) * (S - FLOATF(IB)) * + (S - FLOATF(IC)))
799 WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
601 FORMAT (4H A=,I5,5H B=,I5,5H C=,I5,8H AREA=,F10.2, + 13H SQUARE UNITS)
601 STOP
601 END
What Was Your First Programming Language?
History of Programming Languages

Give a boring lecture on the history of programming languages

In-class exercise
History of Programming Languages

Split up into four groups (randomly)

• **Group 1:** 1960 – 1980
• **Group 2:** 1980 – 1995
• **Group 3:** 1995 – 2010
• **Group 4:** 2010 – Present
History of Programming Languages

1. Spend approximately 20 minutes in your group searching the internet (Google, Wikipedia).

2. Identify:
   • What languages seem important (that you’ve heard of)?
   • What is the “coolest” language?
   • What is the origin of the language?
     • Research (PhD)
     • Industry (IBM, Kodak, HP, Apple, Google, Microsoft, etc)
     • Hobby (someone’s fun project)
   • Trends of languages during the time period
History of Programming Languages

3. Regroup after 20 minutes
4. Have each group give a short presentation with your findings
   • Don’t worry -- I’ll help you out