Lecture 1: Overview and Evaluation Criteria of Programming Languages

CS4080

(Chapter 1 & Chapter 2)

Part 1:
Overview of Programming Languages (PLs)

Why study the “concepts of PLs”?

- Increased ability to express ideas
  - Use appropriate language features in program development
- Improved background for choosing appropriate languages
  - e.g. what language to use for an AI application?
- Increased ability to learn new languages
  - Yes, everyone will learn a new language this semester.
- Better understanding of the significance of implementation
  - How recursion implemented?
  - Which implementation more efficiency?
- Overall advancement of computing
  - History will teach us “a lot of things”.
  - Looking back the languages used in the past you’d much appreciate the ones we’re using now.

Programming Domains

- Scientific applications
  - Large numbers of floating point computations; use of arrays
  - Fortran, C++, C, …
- Business applications
  - Produce reports, use decimal numbers and characters
  - COBOL, SQL, PL/B, Java, C#(.NET) …
- Artificial intelligence
  - Symbols rather than numbers manipulated; use of linked lists
  - LISP, Prolog, C++, …
- Systems programming
  - Need efficiency because of continuous use
  - C, C++, PL/I, GO (google), …
- Web Software
  - Collection of languages: markup (e.g., XHTML), scripting (e.g., PHP), general-purpose (e.g., Java)

Language Categories

- Imperative
  - Central features are variables, assignment statements, and iteration
  - Include languages that support object-oriented programming
  - Include scripting languages
  - Include the visual languages
  - Examples: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Functional
  - Main means of making computations is by applying functions to given parameters
  - Examples: LISP, Scheme
- Logic
  - Rule-based (rules are specified in no particular order)
  - Example: Prolog
- Markup/programming hybrid
  - Markup languages extended to support some programming
  - Examples: JSTL, XSLT

Domain Specific Languages (DSLs)

- A domain-specific language (DSL) is a computer language specialized to a particular application domain. [Wikipedia]
  - General-purpose languages (GPLs): broadly applicable across domains.
- Varieties of DSLs
  - domain-specific markup languages
    - widely used languages for common domains, such as HTML for web pages
  - domain-specific modeling languages
    - specification languages
  - domain-specific programming languages
    - used by only one or a few pieces of software

Domain Specific Languages (DSLs)

- Case Studies
  - P4 is a programming language for controlling packet forwarding planes in networking devices, such as routers and switches
  - Others?
    - Discussion
- The line between general-purpose languages and domain-specific languages is not always sharp
  - Perl: designed for text processing but later used as a GPL
  - postscripts: in principle can be used for any task, but in practice is narrowly used as a page description language.

Language families: contrast

```
//C -- imperative

int gcd (int a, int b) {
  if (a == b) return a;
  else if (a > b)
            return gcd(a-b, b);
  else return (a, b-a);
}
```

```
#Scheme -- functional

(define gcd
   (lambda (a b)
       (cond ((= a b) a)
                 ((> a b) (gcd (- a b) b))
                 (else (gcd a (- b a))))))
```

```
%Prolog – logic
gcd(A,B,G) :- A = B, G=A.
gcd(A,B,G) :- A>B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B>A, C is B-A, gcd(C,A,G).
```

Influences on Language Design

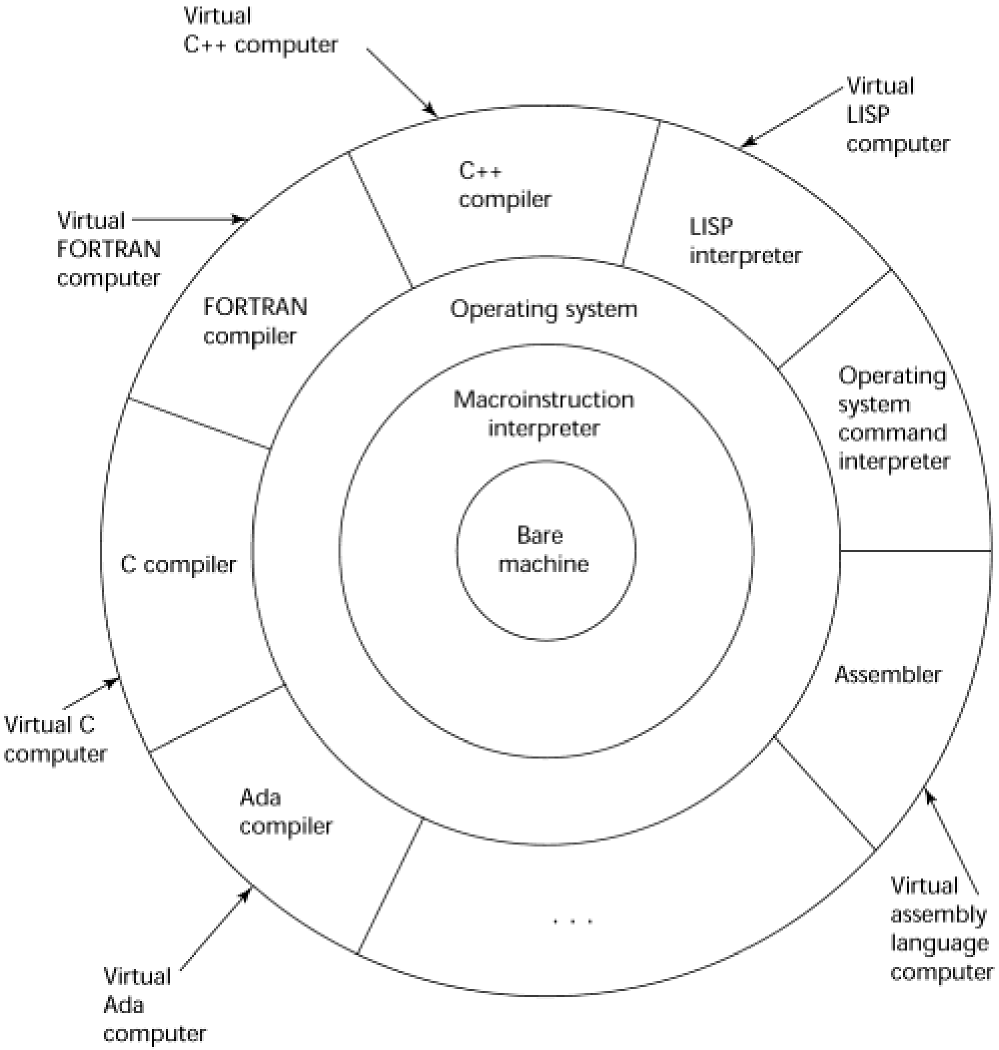
- Computer Architecture
  - von Neumann architecture
    - Stored program concepts
    - Variables model memory cells
  - Parallel computers and multicores
    - Concurrency, multithreading, …
    - Quantum computing
- Programming design methodologies
  - Object-oriented programming
    - Data-oriented vs. procedure oriented

Implementation Methods

- Compilation
  - Translate source code to object/machine code
  - Main modules
    - Syntax analysis, semantics analysis, code generation
  - Executable user code and system code are linked together and loaded to system before execution
    - Linking and loading process
- Pure interpretation
  - A software, called interpreter, acts as a software simulation of a machine
    - Provides a virtual machine for the language
- Hybrid implementation system
  - Translate into intermediate code that allows easy interpretation

Layered View of Computer

The operating system and language implementation are layered over machine interface of a computer



Compilation Process

Figure 1.3

The compilation process

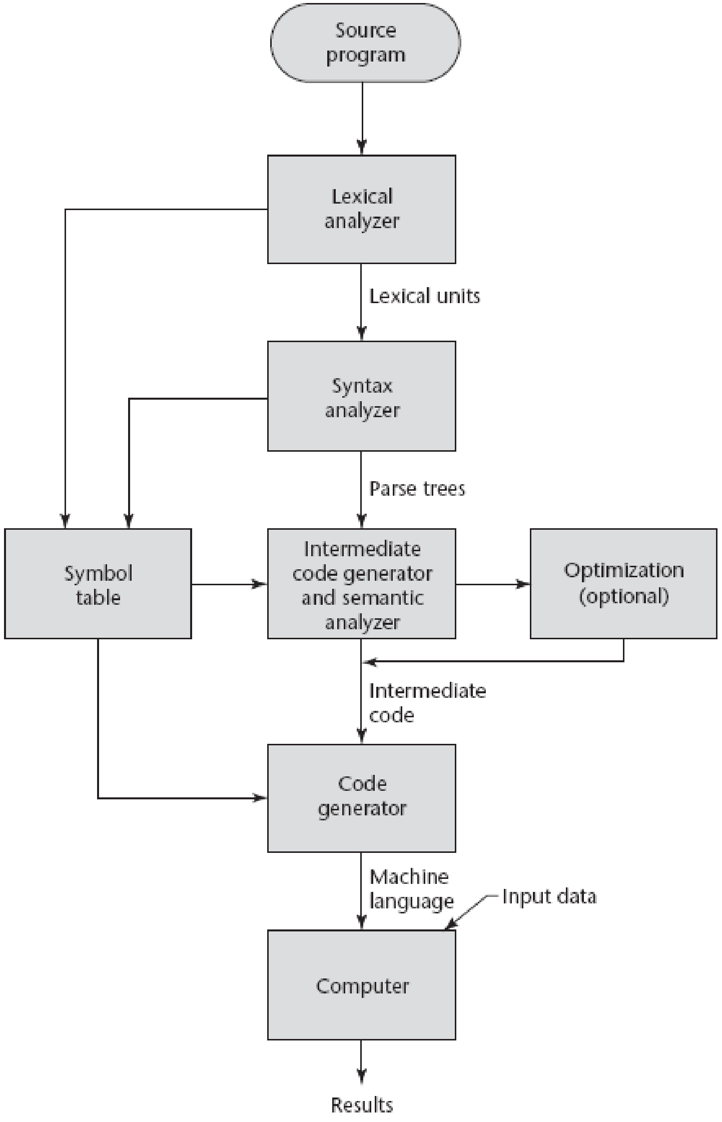
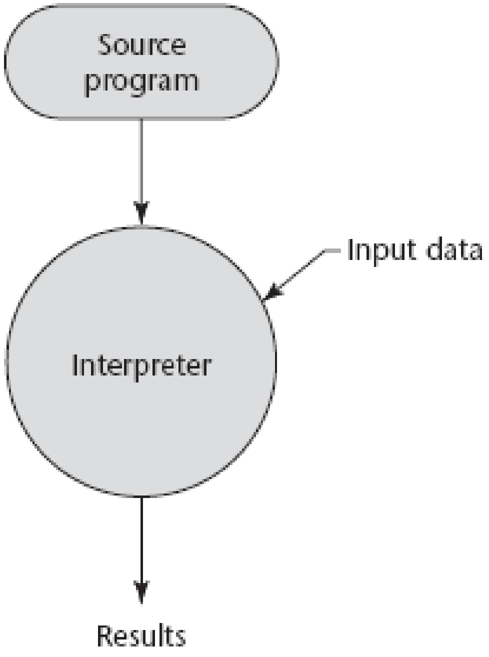


Figure 1.4

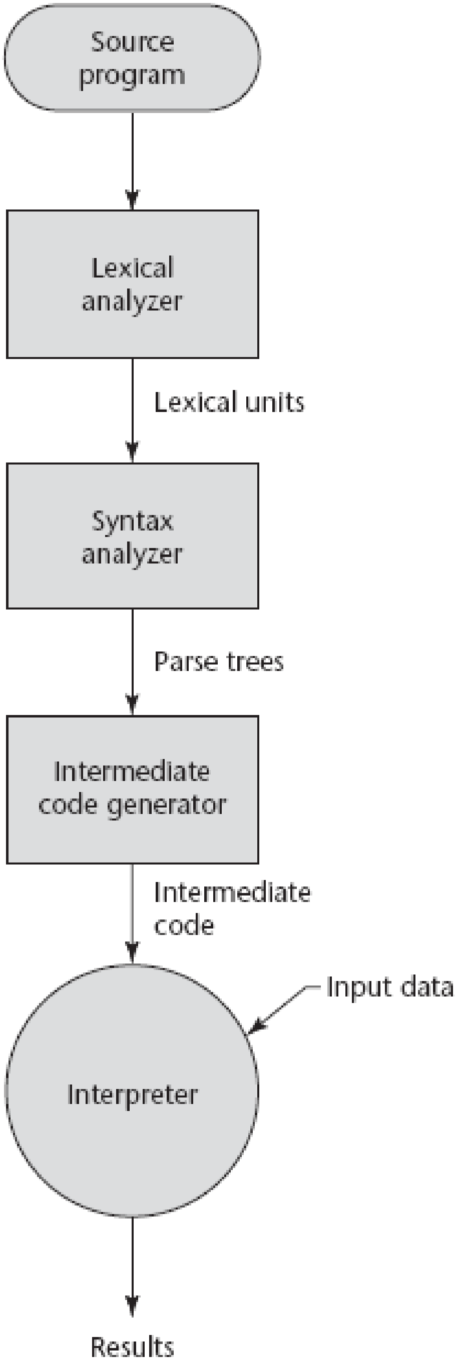
Pure interpretation



Pure Interpretation

Figure 1.5

Hybrid implementation system



Hybrid

Just-in-Time Implementation Systems

- Initially translate programs to an intermediate language
- Then compile the intermediate language of the subprograms into machine code when they are called
- Machine code version is kept for subsequent calls
- JIT systems are widely used for Java programs
- .NET languages are implemented with a JIT system
- In essence, JIT systems are delayed compilers

Programming Environment

- A collection of tools used in the development of software
  - Tools
    - file system, text editor, compiler/linker, …
    - Could also be a large collection of integrated tools
  - Examples
    - Unix environment
    - Borland Jbuilder (for Java development)
    - NetBeans (for Java applications but also support JavaScript, Ruby, and PHP)
    - Microsoft Visual Studio .Net

Genealogy of Common Languages

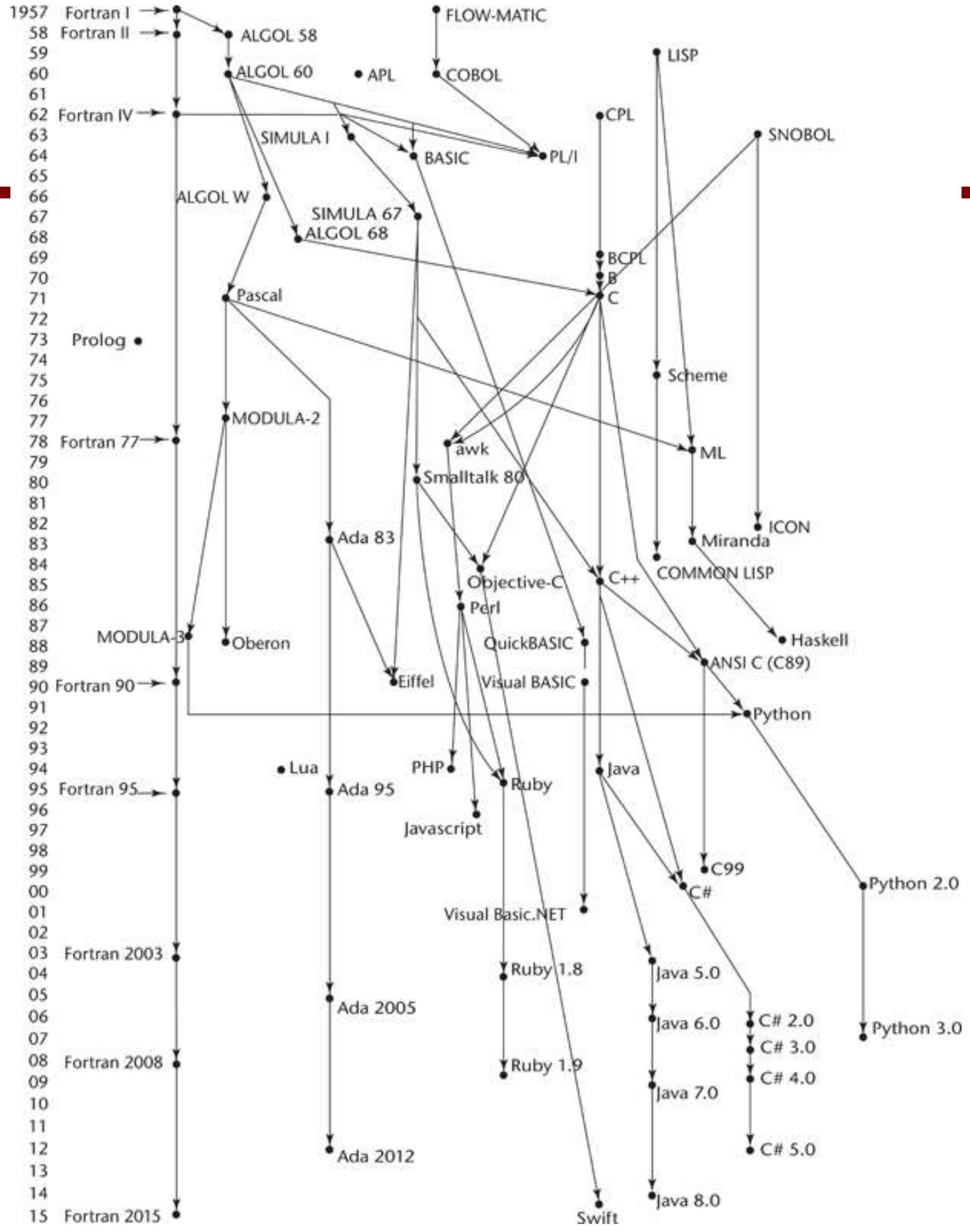


Figure 2.1

Why “high-level” programming languages?

- What was wrong with using machine code?
  - Poor readability
  - Poor modifiability
  - Expression coding was tedious
  - Machine deficiencies--no indexing or floating point
  - Example; x86 instruction set machine code for GCD program

```
55 89 e5 53   83 ec 04 89   e4 f0 e8 31   00 00 00 90   c3 e8 2a 00
00 00 39 c3   74 10 8d b6   00 00 00 00   39 c3 7e 13   29 c3 39 c3
75 f6 89 1c   24 e8 6e 00   00 00 8b 5d   fc c9  c3 29   d8 eb eb 90
```

Machine code, assembly, high-level programs

- GCD program in x86 assembly language
  - The first few lines of the GCD assembly code …

```
pushl   %ebp
movl    %esp, %ebp
pushl   %ebx
subl    $4, %esp
andl    $-16, %esp
…
```

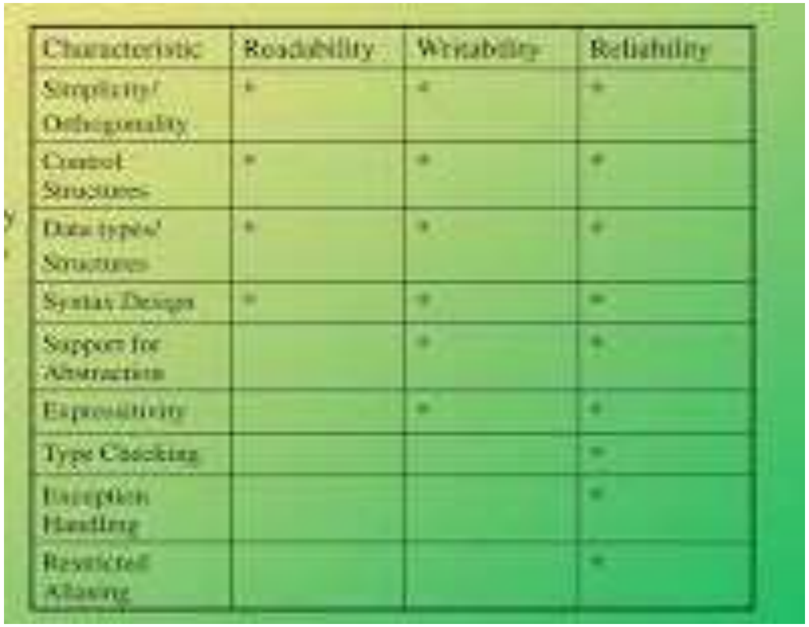
- Can you write a GCD program in Java/C++/Python?

What could we learn from the history?

- Understand obscure features
  - E.g. C++: union type, multiple inheritance, * operator (in pointers) …
  - Why they're gone?
- Choose among alternative ways to express things
  - E.g. copy constructor vs. extra assignment
- Simulate useful features in languages that lack them
  - E.g. iterator
- Make better use of technology
  - Web-based language such as XML, etc.
- …

Part 1: Learning Objectives

- After studying the Part 1 of Lecture 1, you should be able to
  - Describe the history and development of programming language design
    - Discuss what could we learn from the history?
  - Classify Languages based on language features and application domains
    - Language categories
  - Discuss the perspectives and current issues in language design



Characteristic	Readability	Writability	Reliability
Simplicity/ Orthogonality	*	*	*
Control Structures	*	*	*
Data types/ Structures	*	*	*
Syntax Design	*	*	*
Support for Abstraction		*	*
Expressivity		*	*
Type Checking			*
Exception Handling			*
Restricted Aliasing			*

Part 2:
Language Evaluation Criteria

Which programming language is the “best”?

- No “best”, but look for “better”
- Programming domain specific
  - E.g. Business vs. System programming
- Bad design or good design?
  - Many design choices
    - E.g. case sensitive vs. case insensitive
  - Not wrong or right, but which one is better?
- Language features are important
  - Languages of different category may offer different features
    - E.g. functional languages vs. mark-up languages
  - We will focus on feature comparisons of similar languages
    - E.g. lists. vs. arrays
    - E.g. should we have a switch statement or not?

Language Evaluation Criteria

- **Readability**: the ease with which programs can be read and understood
  - Syntax, data types, …
- **Writability**: the ease with which a language can be used to create programs
  - Support for abstraction, expressivity (operators, predefined functions, …)
- **Reliability**: conformance to specifications (i.e., performs to its specifications)
  - Type checking, exception handling, …
- **Cost**: the ultimate total cost
  - Training programmers, compiling/executing programs, implementation, …
- Portability, generality, …

Which one has better readability?

```
//C:

int gcd (int a, int b) {
  if (a == b) return a;
  else if (a > b)
            return gcd(a-b, b);
         else return (a, b-a);
}
```

```
#Scheme:

(define gcd
   (lambda (a b)
       (cond ((= a b) a)
                 ((> a b) (gcd (- a b) b))
                 (else (gcd a (- b a))))))
```

```
%Prolog
gcd(A,B,G) :- A = B, G=A.
gcd(A,B,G) :- A>B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B>A, C is B-A, gcd(C,A,G).
```

Answers may be subjective, but here, the ”readability” criterion attempts to set a common ground.

Which one has better writability?

```
//C:

int gcd (int a, int b) {
  if (a == b) return a;
  else if (a > b)
            return gcd(a-b, b);
        else return (a, b-a);
}
```

```
#Scheme:

(define gcd
  (lambda (a b)
     (cond ((= a b) a)
            ((> a b) (gcd (- a b) b))
            (else (gcd a (- b a))))))
```

```
%Prolog
gcd(A,B,G) :- A = B, G=A.
gcd(A,B,G) :- A>B, C is A-B, gcd(C,B,G).
gcd(A,B,G) :- B>A, C is B-A, gcd(C,A,G).
```

Discussion Questions (which? why?)

- Which one has better writability?
  - Java, JavaScript, Ruby, Python, …?
- Which one is more reliable?
  - Java, C++, C#, Ada, …?
- Which one has better (execution) performance?
  - Java, C, Python, Ruby, …?
- Which language is the best to learn as the 1st programming language?
  - Java, Python, Visual Basic, …
- Assume you took CS4110 and you're assigned to write a compiler for a PL, which task is the easiest?
  - Java, FORTRAN, Ada, Pascal, …

Readability & Writability: which one better?

```
if (ans == ‘A’) {
    … call fa();
}
else if (ans == ‘B’) {
     … call fb();
}
else if (ans == ‘C’) {
     … call fc();
}
else {
     … call ff();
}
```

```
switch (ans) {
case ‘A’ : fa(); break;
case ‘B’ : fb(); break;
case ‘C’ : fc(); break;
default:  ff();
}
```

Readability & Writability: which one better?

```
if (ans == ‘A’) {
    … call fa();
}
else if (ans == ‘B’) {
    … call fb();
}
else if (ans == ‘C’) {
    … call fc();
}
else {
    … call ff();
}
```

```
if ans == ‘A’ :
      … call fa();
elif ans == ‘B’ :
      … call fb();
elif  ans == ‘C’ :
      … call fc();
else :
      …call ff();
```

Feature comparison

- Some language features embedded/inherent in coding
- Java vs. C++

  array index checking

  which one has better reliability?

  which one costs more?

```
//Java – observe the execution
int[] a = new int[5];
for (int i=1; i<=5; i++)
    a[i] = 100;
```

```
//C++ – observe the execution
int a [5]; //or int *a  = new int[5];
for (int i=1; i<=5; i++)
    a[i] = 100;
```

Introducing a new feature

- Example

980_000_000   a valid Java numeric literal?

if no, why Java avoids such format?
if yes, why Java supports such format?

98_   _98   valid?

Language Design Trade-Offs

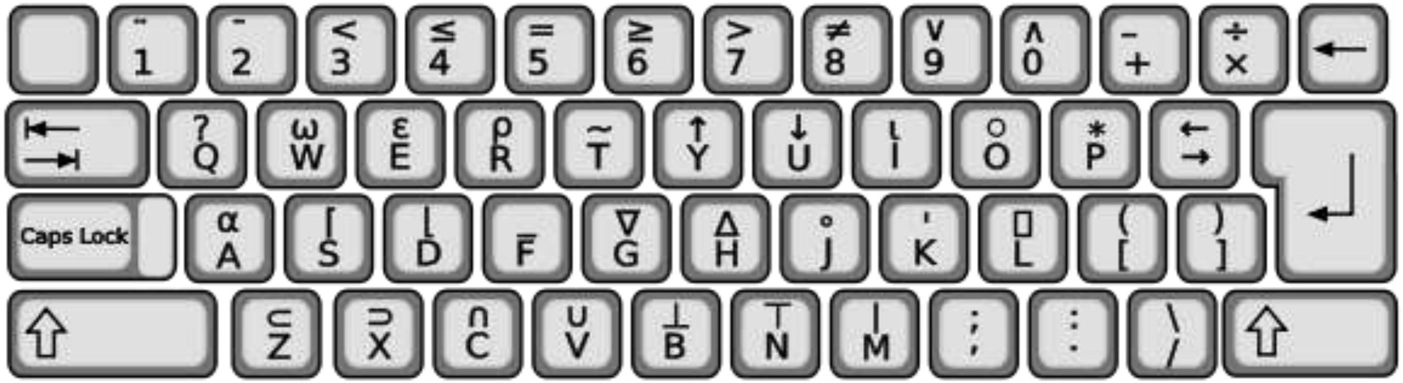
- Reliability vs. cost of execution
  - Example: Java demands all references to array elements be checked for proper indexing, which leads to increased execution costs

- Readability vs. writability

  Example: APL provides many powerful operators (and a large number of new symbols), allowing complex computations to be written in a compact program but at the cost of poor readability (attachment: APL keyboard image)

- Writability (flexibility) vs. reliability
  - Example: C++ pointers are powerful and very flexible but are unreliable

APL keyboard



Discussion question: Pros and Cons of APL’s design philosophy?

APL references:

https://en.wikipedia.org/wiki/APL_(programming_language)
https://computerhistory.org/blog/the-apl-programming-language-source-code/

Pitfalls in applying criteria

- How Subjective/Objective?

- Which of the following codes you like more?
- Or, which of the following codes you think better?

```
//Java or C++
if (a < b) {
        sum  += a;
        less++
}
```

```
#Python
if a < b :
        sum += a
        less += 1
```

Restating the question:

(a) Which code has better readability?

(b) Which code has better writability?

Will that change your vote?

Which Criterion more important?

- Is “readability” more important than “writability”?
- Should the “cost” the No. 1 criterion?
- Should we enforce high “reliability” in every language?
- Are “portability” or “generality” less important?

Answer may vary with specific applications/usages, …

In this course, we focus on the criteria “readability”, “writability”, and “reliability” equally, also with some consideration to “cost”.

Learning Objectives

- After studying the Part 2 of Lecture 1, you should be able to:
  - Apply a set of language evaluation criteria such as readability, writability, … in an attempt to provide a non-biased, objective way of evaluation
  - Describe and discuss the trade-offs among the evaluation criteria

Activity 1
Introduction of Group’s Language

- This is a group-based assignment.
  - Each of you should join a language group.
  - If you don’t have a group yet, please contact me.
  - Each group will make a presentation (3-5 minutes) to introduce your language.
- Activity 1 covers two parts
  - Part A: Form groups
  - Part B: Language introduction

Part B of Activity 1: Language Introduction

- History, Origin, and any fun facts about the language
- Suggested Part A presentation topics:
  - Who designed this language?
  - What is the design goal of this language?
  - What languages are its ancestors?
  - What it differs from its ancestors?
    - What it learned from the history?
    - In which way your language is better than its ancestors?
  - What new feature (by the time of its first publication) did this language introduce, if any?
  - Why are you interested in this language?
  - How widely used is the language?

Don’t have to answer all the questions exactly as listed.

Feel free to introduce your language in your own way.

Java Group – short version

- Members: Lan & Charlotte
- Java: originally developed by James Gosling at Sun Microsystems

  - Initially to meet a need for a reliable language for communication in embedded devices
  - first released in 1995 as a general purpose language
  - Now acquired and maintained by Oracle
- Strongly influenced by C/C++
  - Java is more reliable than its ancestors, and
  - Better OOP features
- Interested in learning Java
  - Java is hot!
  - Popularly used in industry.

Part B of Activity 1: Language's syntax definition

details to be discussed next lecture

- Find a site with formal syntax/grammar description for your language (note: there may be multiple sites, try to find one with EBNF like description.)
  - e.g. (assume I'm the Java group), the site I found is:

https://docs.oracle.com/javase/specs/jls/se13/html/index.html

Activity A1 – Deliverables

- Presentation
  - Penalty taken for absences
  - One-time absence may be excused given for reasonable (and prior unless emergency) justification
- Slides submission on Canvas assignment
- Due date(s): see Canvas