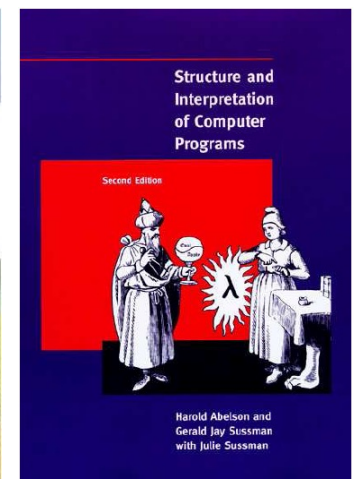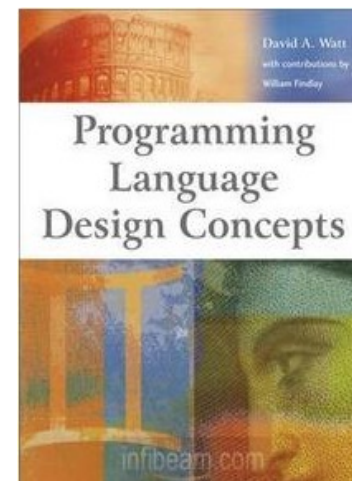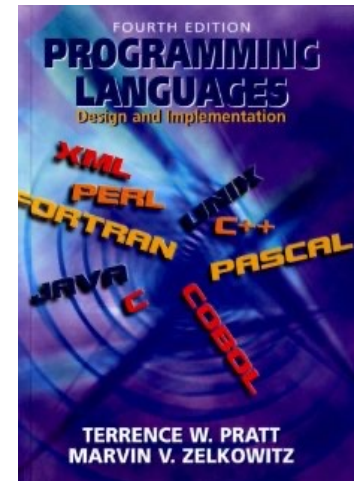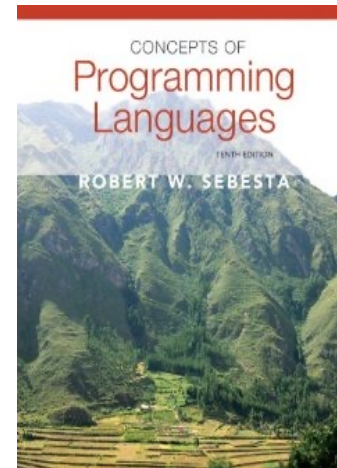# CS391: Principles of Programming Languages

Dr. Raju Halder

References

1. Robert W. Sebesta, "**Concepts of Programming Languages**", Addison-Wesley.

2. T.W. Pratt and M.V. Zelkowitz, "**Programming Languages – Design and Implementation**", Prentice-Hall.

3. D. A. Watt, **"Programming Language Design Concepts"**, John Wiley & Sons.

4. Harold Abelson, Gerald Jay Sussman, Julie Sussman, **"Structure and Interpretation of Computer Programs"**, MIT Press, 2nd Edition.

References

5. Ravi Sethi, "**Programming Languages - Concepts and Constructs**", Addison-Wesley.

6. Gilles_Dowek, **"Principles of Programming Languages",** Springer.

7. Michael L. Scott, **"Programming Language Pragmatics"**, Morgan Kaufmann

8. Kenneth C. Louden and Kennath A. Lambert, **"Programming Languages: Principles and Practice"**, Cengage Learning.
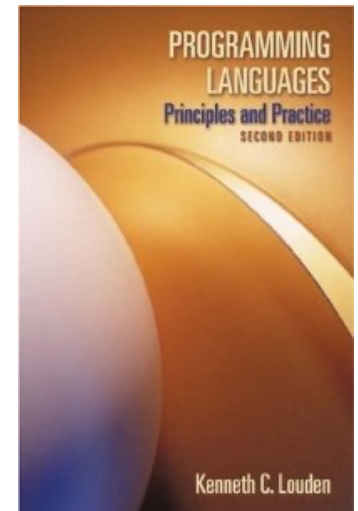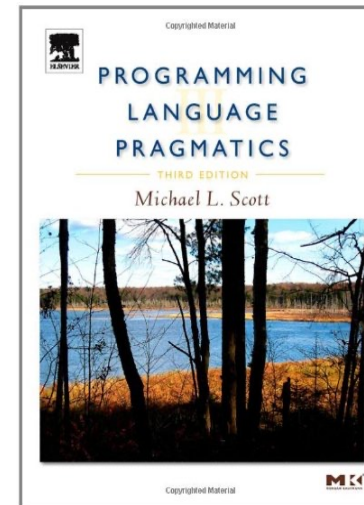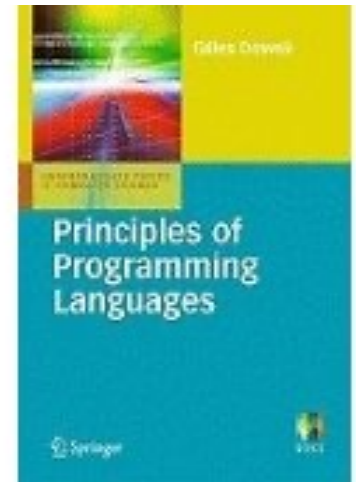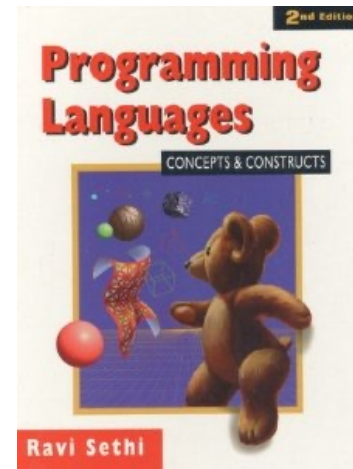
References

5. A. B. Webber, "**Modern Programming Languages: A Practical Introduction**", Franklin Beedle & Associates.
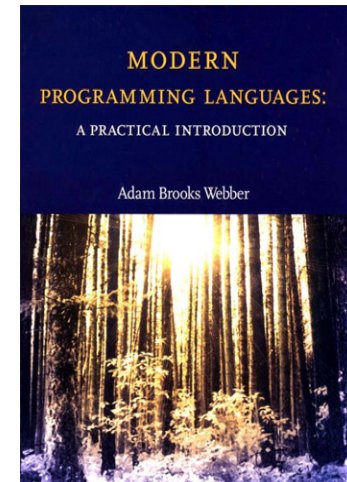
CS391: Principles of Programming Languages

Visit:

- http://172.16.1.3/~halder/Programming_Languages/

Contact Us:

- halder@iitp.ac.in

Meeting Time:

- Anytime

CS391: Principles of Programming Languages

- ## Teaching Assistants
  - [Ph.D.] Imran Alam (imran.pcs16@iitp.ac.in)
  - [Ph.D.] Akshay M. Fajge (fajge_1921cs12@iitp.ac.in)
  - [Ph.D.] Swagatike Sahoo (swagatika_1921cs03@iitp.ac.in)

Topics to Cover

- Why do we study Principles or Programming Languages (PPL)?
- Why so many programming languages? What to choose?
- Different Programming Language Paradigms.
- How to define Syntax and Semantics of PL?
- Introduction to PL translators: Compiler and Interpreter
- Type Systems
- Binding and Scoping
- Memory Management
- Introduction to Imperative PL, OO PL, Functional PL, Logic PL

What is a Programming Language?

What is a Programming Language?

- A **<u>programming language</u>** is a notational system for describing computation in machine-readable and human-readable form.

❑ According to Stroustrup, a programming language is
   - ❖ a tool for instructing machines,
   - ❖ a means for communicating between programmers,
   - ❖ a vehicle for expressing high-level designs,
   - ❖ a notation for algorithms,
   - ❖ a way of expressing relationships between concepts,
   - ❖ a tool for experimentation,
   - ❖ a means for controlling computerized devices.

- "My view is that a general purpose programming language must be all of those to serve its diverse set of users" – Bjarne Stroustrup

Levels of Programming Languages

- **<u>high-level languages</u>** A language is higher level if it is human-readable and independent of the underlying machines.
- **<u>low-level languages</u>**. Low-level languages are designed by introducing names and symbols at the place of actual codes for m/c operations, values, and storage locations, making them more readable. Assembly language is a low-level languages
- **<u>machine Languages</u>** An unintelligible code that is only understood by the computer.

Levels of Programming Languages

High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

Low-level program

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Executable Machine code

```
000100100100010100100 1
00111011001010110100 1.
..
```

Towards Higher Level Languages

Towards Higher Level Languages

❑ **Higher-level languages have replaced machine language and assembly language in virtually all areas of programming, because they provide benefits like the following:**

- ❖ Readable, familiar notations
- ❖ Machine independence (portability)
- ❖ Availability of program libraries
- ❖ Consistency checks during implementation that can detect errors

Art of Language Design

- Why are there so many programming languages?

- What are the properties of programming languages?

- What makes a language successful?

- Why do we need programming languages?

Why so many programming languages?

Today there are thousands of high-level programming languages, and new ones continue to emerge.

- **Evolution**
  - We've learnt better ways of doing things over time . E.g., from BASIC to JAVA
- **Orientation toward special purposes**
  - C is good for low-level system programming, whereas AWK is good for manipulating strings.
- **Orientation toward personal preference**
  - Some people like to work with pointers of C, whereas some people like implicit dereferencing of JAVA.

Properties

- A PL must be **universal** – capable of expressing any computation.
    - A language without iteration or recursion is not universal.
    - A language of recursive functions (and nothing else) is universal.
- A PL should be reasonably *natural* for expressing computations in its intended application area.
- A PL must be **implementable** – it must be possible to run every program on a computer.

What makes a language successful?

- **Easy to learn**
- **Expressive Power**
  - Language features clearly have a huge impact on the programmer's ability to write clear, concise, and maintainable code, specially for very large system.
- **Easy to implement**
- **Excellent Compilers**
- **Cost of Use:** translation, testing, execution, maintenance, etc.

TIOBE Programming Community index

- The TIOBE Programming Community index is an indicator of the popularity of programming languages.

- https://www.tiobe.com/tiobe-index/

- The ratings are based on:
  - the number of skilled engineers world-wide, courses and third party vendors.
  - Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings.
  - It is important to note that the TIOBE index is not about the best programming language or the language in which most lines of code have been written.

Dec 2023	Dec 2022	Change		Programming Language	Ratings	Change
1	1			Python	13.86%	-2.80%
2	2			C	11.44%	-5.12%
3	3			C++	10.01%	-1.92%
4	4			Java	7.99%	-3.83%
5	5			C#	7.30%	+2.38%
6	7	^		JavaScript	2.90%	-0.30%
7	10	^		PHP	2.01%	+0.39%
8	6	v		Visual Basic	1.82%	-2.12%
9	8	v		SQL	1.61%	-0.61%
10	9	v		Assembly language	1.11%	-0.76%

TIOBE Programming Community Index

Source: www.tiobe.com

Python · C · C++ · Java · C# · JavaScript · PHP · Visual Basic · SQL · Assembly language
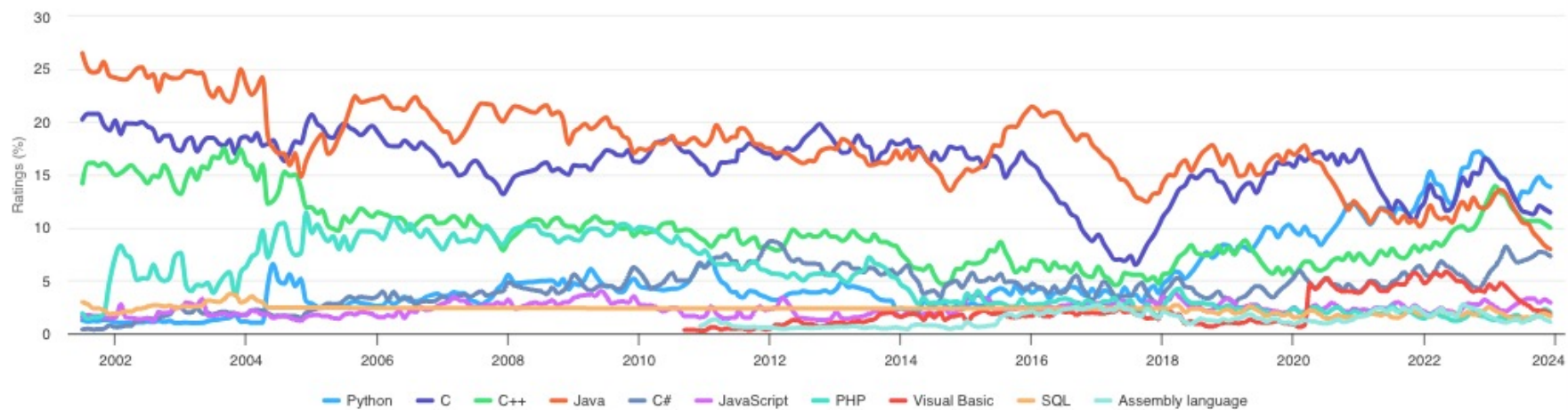
Why study Programming Language Concepts?

1. Increases the capacity to express ideas
2. Better use of existing languages
3. Better choice of programming languages
4. Better understanding of the implementation of concepts
5. Increases ability to learn/design new languages
6. Overall advancement of computing

Why study Programming Language Concepts?

1. Increases the capacity to express ideas

Increases the ability to express ideas

- The depth at which people can think is heavily influenced by the expressive power of their language.

- In searching for data and program structures suitable to the solution of a problem, one tends to think only of structures that are immediately expressible in the languages with which one is familiar.

- Expressing ideas as Algorithms

Why study Programming Language Concepts?

2. Better use of existing languages

Better use of existing languages

- It is uncommon for a programmer to be familiar with and use all of the features of the existing languages, because of their complexities.

- By studying the concepts of programming languages, programmers can learn about previously unknown and unused features of the languages they already used and begin to use those additional features.

Why study Programming Language Concepts?

3. Better choice of programming languages

Better choice of programming languages

- To allow a better choice of programming language

- Some languages are better for some jobs than others. Help you to choose a better language
  - **C** vs Modula-3 vs C++ for systems programming
  - **Fortran** vs APL vs Ada for numerical computations
  - C vs **Ada** vs Modula-2 for embedded systems
  - **Common Lisp** vs Scheme vs ML for symbolic data manipulation
  - **Java** vs C for networked PC programs

- However, people are more likely to use languages with which they are most comfortable than the most suitable one for a particular job.

Why study Programming Language Concepts?

4. Better understanding of the implementation of concepts

Better understanding of significance of implementation

- By understanding how features in your language are implemented, you greatly increase your ability to write efficient program.

- For instance, programmers knowing little about complexity of the implementation of subprogram calls often do not realize that small subprogram that is frequently called can be highly inefficient.

- For instance,
  - understanding how data such as arrays, strings, lists, or records are created and manipulated by your language,
  - knowing the implementation details of recursion, or
  - understanding how object classes are built

  allows you to build more efficient programs consisting

  of such components.

Better understanding of significance of implementation

- It allows us to visualize how a computer executes various language constructs.

- Certain kinds of program bugs can be found and fixed only by a programmer who knows some related implementation details.

Why study Programming Language Concepts?

5. Increases ability to learn/design new languages

Increases ability to learn/design new languages

- Computer science is a relatively young discipline and most software technologies (design methodology, software development, and programming languages) are not yet mature.
  - Therefore, they are still evolving.

- A thorough understanding of programming language design and implementation makes it easier to learn new languages.

- Some languages are similar; easy to walk down the family tree

- It is easier to learn a new language if you understand the underlying structures of language.
    Examples:
    - It is easier for a BASIC programmer to learn FORTRAN than C.
    - It is easier for a C++ programmer to learn Java.
    - It is easier for a Scheme programmer to learn LISP.

Why study Programming Language Concepts?

6. Overall advancement of computing

Overall advancement of computing

- There is a global view of computing that can justify the study of programming Language concepts.

- Frequently, the most popular language may not be the best language available.

- E.g., ALGOL 60 did NOT displace Fortran.
  - Programmers faced difficulty in understanding its description and they didn't see the significance of its block structure and well-structured control statements until many years later.

Programming Language Concepts

- **Concepts** are building blocks of programs and PLs:
  - values and types
  - variables and storage
  - bindings and scope
  - procedural abstraction
  - data abstraction
  - generic abstraction
  - concurrency.

The Amazing Variety

- How many Computer Languages are there?
  - **late 1940s** first electronic computers & LLLs
  - **1950s** first HLLs for computers
  - **1969** about 120 HLLs, about 15 in widespread use
  - **1977** about 80 HLLs in active (non-trivial) use
  - **Today** more than 2000 HLLs
- A **paradigm** is a style of programming, characterized by a particular selection of key concepts.
  - Imperative
  - Functional
  - Logic
  - Object-oriented

Programming Paradigms

- Imperative
  - examples: C, Pascal, Basic, Fortran
- Functional
  - examples: Lisp, ML
- Rule-based (or Logic)
  - example: Prolog
- Object-oriented
  - examples: C++, Java, Smalltalk

Imperative Languages

- Example: a factorial function in C

$$factorial\ (n) = \begin{cases} 1 & n = 0 \\ n*factorial\ (n-1) & n > 1 \end{cases}$$

```
int factorial (int n) {
int i, fact=1;
    for (i=1; i<=n; i++)
        fact = fact * i;
    return fact;
}
```

Imperative Languages

Hallmarks:

- Statement oriented languages
- Every statement changes the machine state
- Computation is expressed by a sequence of actions: Order of execution is critical
- Heavily based on von Neumann architecture
- Examples: Fortran, C

Functional Languages

- Example: a factorial function in ML

```
fun fact (x) =
  if x = 0 then 1 else x * fact(x-1);
```

- Example: a factorial function in Lisp

```
(defun fact (x)
  (if (= x 0) 1 (* x (fact (- x 1)))))
```

- Hallmarks of functional languages:
  - Single-valued variables
  - Heavy use of recursion

Logic Languages

- Example: a factorial function in Prolog

```
fact(X,1) :-
  X == 1.
fact(X,Fact) :-
  X > 1,
  NewX is X - 1,
  fact(NewX,NF),
  Fact is X * NF.
```

- Hallmark of logic languages
  - Program expressed as rules in formal logic

Declarative

- Problem specification using logic or functions.

- Both functional programming and Logic programming are declarative.

- Examples: LISP, ML, Haskell, Prolog

Object-Oriented Languages

Example: a factorial function in C++

```
public class MyInt {
private int value;
public MyInt(int value) {
  this.value = value;
}
public int getValue() {
  return value;
}
public MyInt getFact() {
  return new MyInt(fact(value));
}
private int fact(int n) {
  int sofar = 1;
  while (n > 1) sofar *= n--;
  return sofar;
}
}
```

Object-Oriented Languages

Hallmarks:

- Constructs to help programmers use "objects"

- Based on the concept of data abstraction.

- Uses encapsulation (data hiding)

- Supports inheritance and polymorphism

- Suitable for programming in the large.

- Examples: C++, Java, Smalltalk, Eiffel

Programming Paradigms

❑ **Programming paradigms are ways of thinking about programming.**

❑ *Imperative Programming*

- ❖ Imperative languages are action oriented; that is, a computation is viewed as a sequence of actions.
- ❖ actions take effect by modifying some memory space (denoted by variables)
- ❖ E.g., Algol, Pascal, C, etc.

❑ *Functional Programming*

- ❖ Simply put, functional programming is programming without assignments.
- ❖ To mimic mathematical functions to the greatest extent
- ❖ E.g., Lisp, Scheme, ML, etc.

Programming Paradigms

❑ *Object-Oriented Programming*
  - ❖ Central to object-oriented programming is the concept of objects and their classification into classes and subclasses.
  - ❖ E.g., Smalltalk, C++, Java, etc.

❑ *Logic Programming*
 - ❖ A kind of rule-based programming.
  - ❖ E.g., Prolog

Programming Paradigms

|	
Imperative style:	program = algorithms + data *good for decomposition*
Functional style:	program = functions $\circ$ functions *good for reasoning*
Logic programming style:	program = facts + rules *good for searching*
Object-oriented style:	program = objects + messages *good for modeling*

A Brief Chronology

| Early 1950s		*"order codes" (primitive assemblers)*
1957	FORTRAN	*the first high-level programming language*
1958	ALGOL	*the first modern, imperative language*
1960	LISP, COBOL	*Interactive programming; business programming*
1962	APL, SIMULA	*the birth of OOP (SIMULA)*
1964	BASIC, PL/I	
1966	ISWIM	*first modern functional language (a proposal)*
1970	Prolog	*logic programming is born*
1972	C	*the systems programming language*
1975	Pascal, Scheme	*two teaching languages*
1978	CSP	*Concurrency matures*
1978	FP	*Backus' proposal*
1983	Smalltalk-80, Ada	*OOP is reinvented*
1984	Standard ML	*FP becomes mainstream (?)*
1986	C++, Eiffel	*OOP is reinvented (again)*
1988	CLOS, Oberon, Mathematica	
1990	Haskell	*FP is reinvented*
1990s	Perl, Python, Ruby, JavaScript	*Scripting languages become mainstream*
1995	Java	*OOP is reinvented for the internet*
2000	C#	

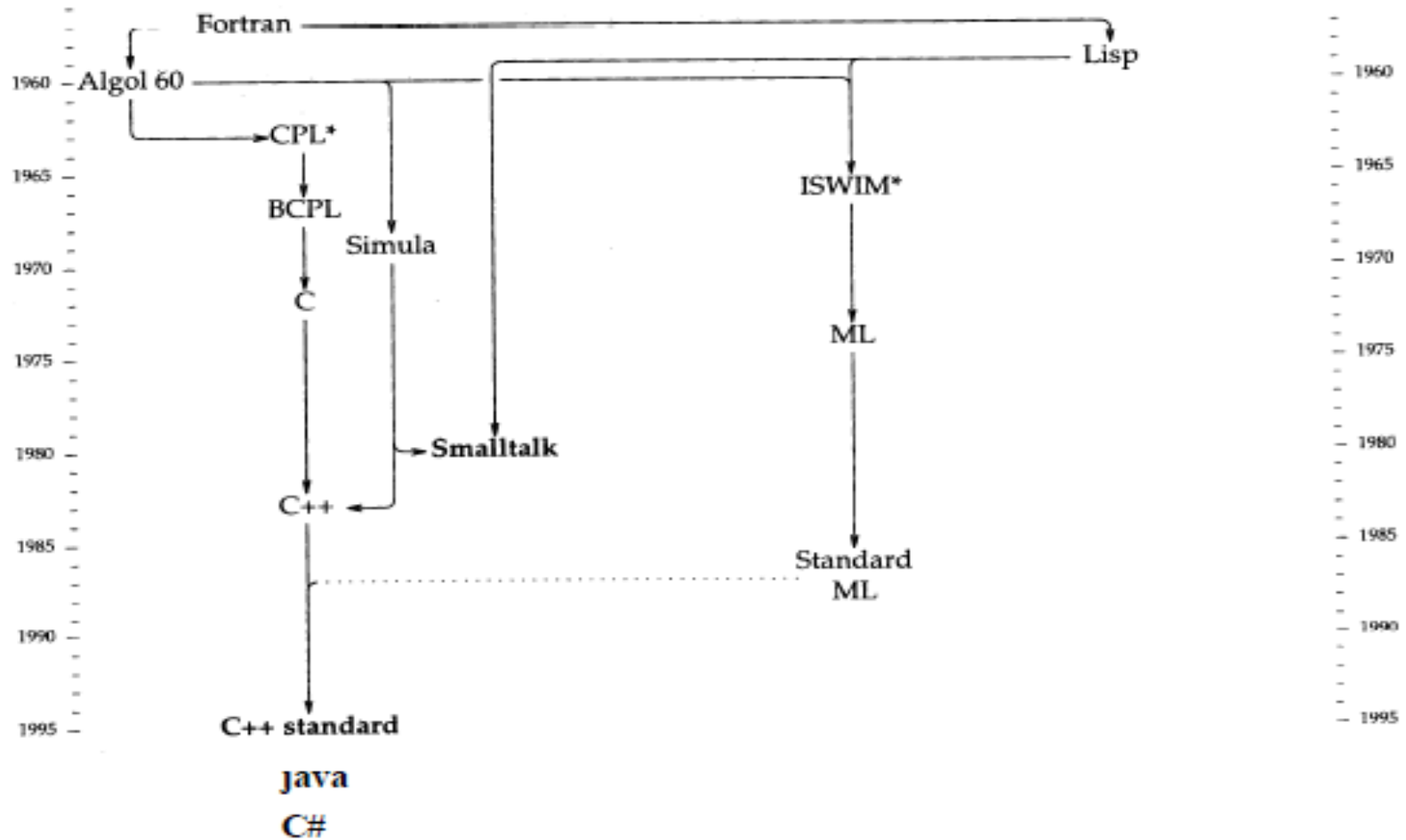# Dates and ancestry of major programming languages
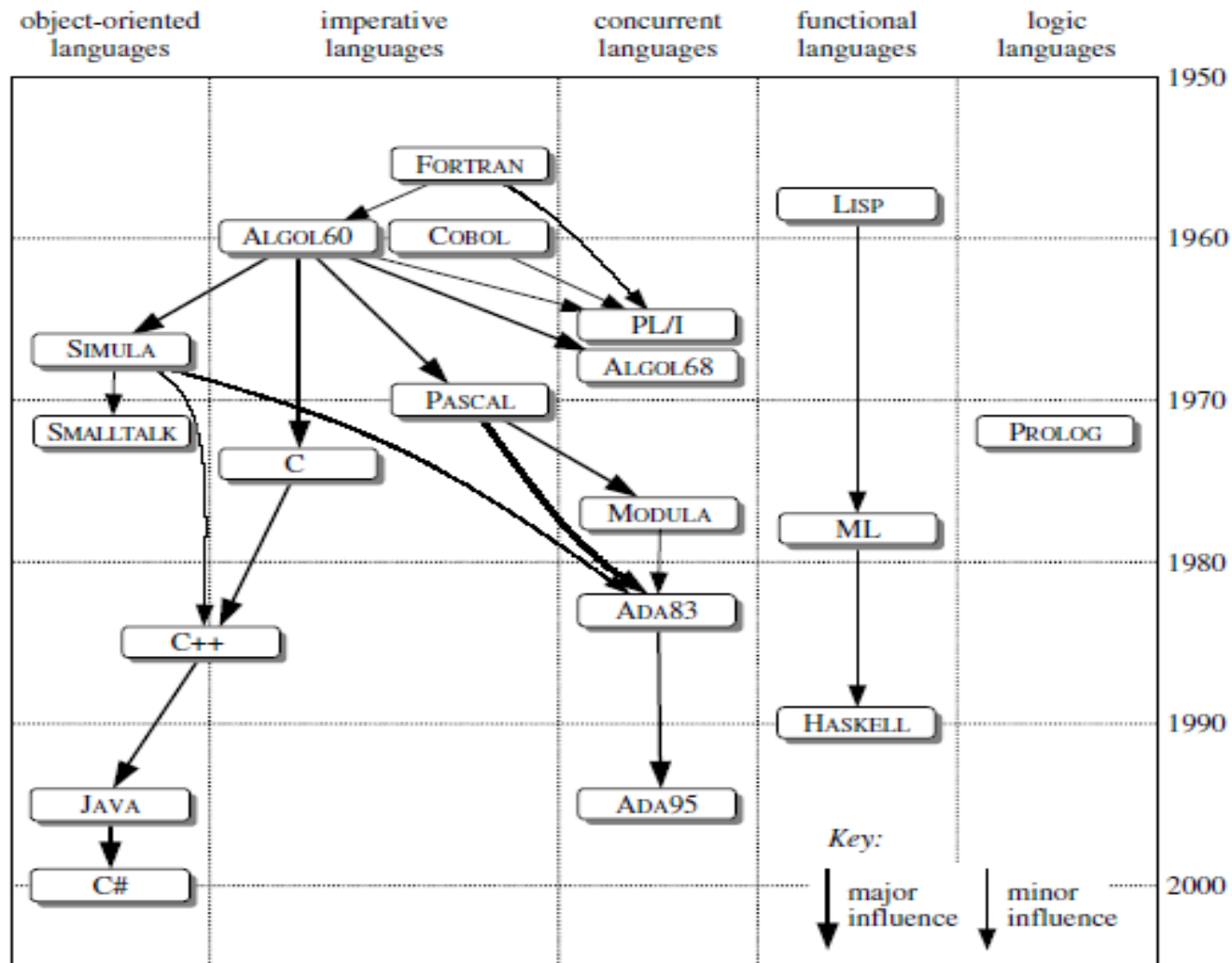
The Imperative Family
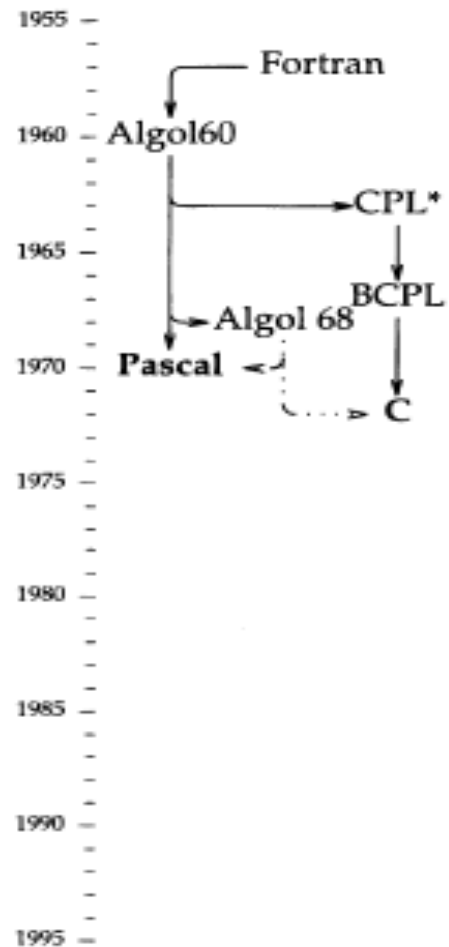
The Functional Family
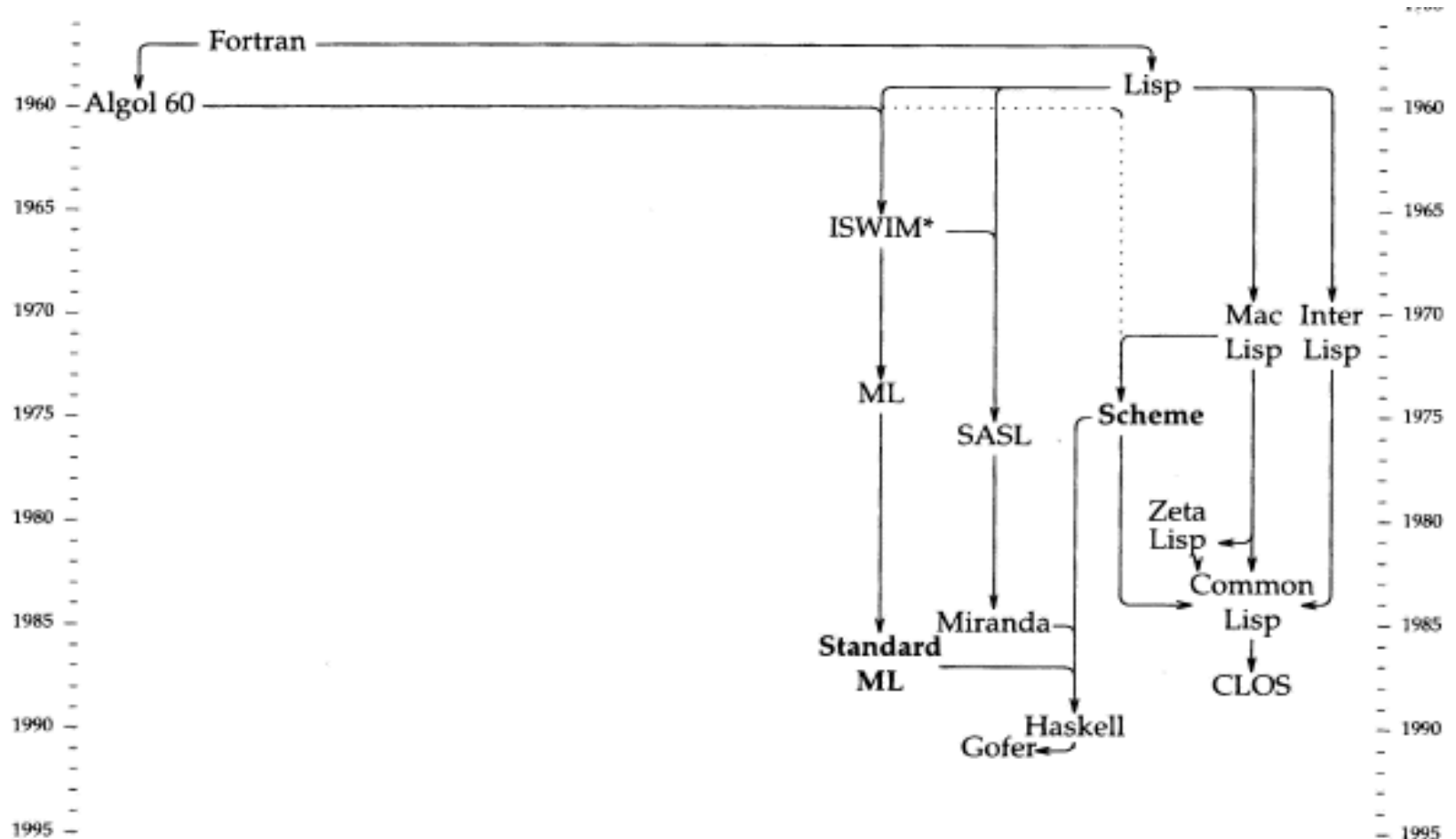
The Object-Oriented Family

Programming Domains

- <u>Scientific</u>: Heavily based on numerical algorithms (Fortran, C)

- <u>Business Applications</u>: Storage, retrieval, and formatting of data, reporting. (COBOL)

- <u>Artificial Intelligence</u>: Symbolic computing, List directed processing (LISP, Prolog)

- <u>Systems Programming</u>: Fast, Low level features (C)

- <u>Internet</u>: Web based programming (Perl, Java)

- <u>Simulation</u>: Process modeling (MATLAB, GPSS)

Networking Era

❏ **A goal in the late 1980s was to make the retrieval of information easy. The breakthrough came in 1989 at CERN, for the development of WWW and HTML.**

❏ **The Web poses new issues to programming languages:**
- ❖security
- ❖performance
- ❖platform independence

❏ **Some successful achievements are Java, HTML, and XML.**

❏ **Script languages to add dynamic t Web pages**
- ❖JavaScript: client-side, HTML-embedded scripting language
- ❖PHP: server-side, HTML-embedded scripting language
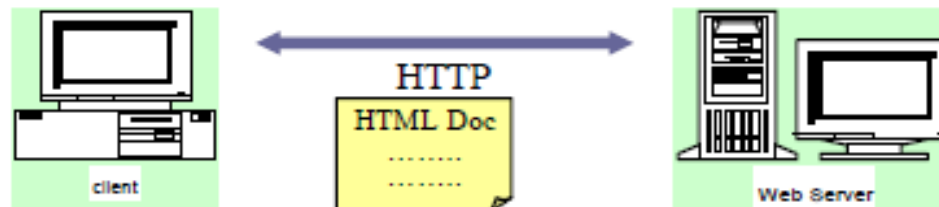
Concurrent

- Parallel execution of processes.

- Multi-tasking or multi-threading primitives.

- Inter process communication and synchronization.

- Helps in speeding up the execution of parallel algorithms

- Examples: Concurrent C, Java, Fortran-90