Concepts of Programming Languages

Lecture 2 - History of Programming Languages

Patrick Donnelly

Montana State University

Spring 2014

Administrivia

Website:

`http://nisl.cs.montana.edu/~pdonnelly/CSCI305/`

Reading:

Chapter 1

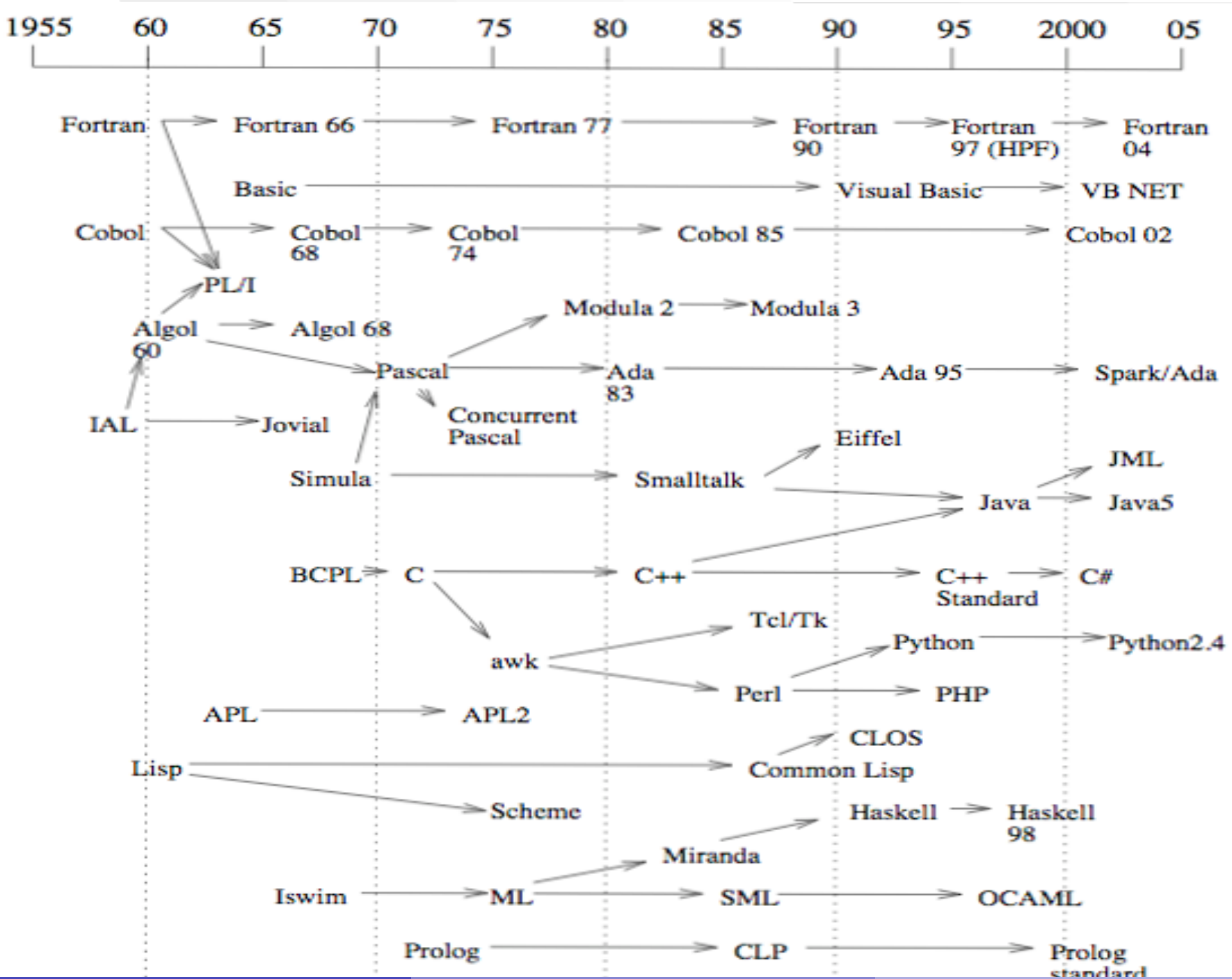
A good programming language is a conceptual universe for thinking about programming.

A. Perlis

Programming Domains

The following user communities claim major developments in programming languages:

- Artificial Intelligence
- Computer Science Education
- Science and Engineering
- Information Systems
- Systems and Networks
- World Wide Web


1955
60
65
70
75
80
85
90
95
2000
05
Fortran
Fortran 66
Fortran 77
Fortran 90
Fortran 97 (HPF)
Fortran 04
Basic
Visual Basic
VB NET
Cobol
Cobol 68
Cobol 74
Cobol 85
Cobol 02
PL/I
Algol 60
Algol 68
Modula 2
Modula 3
Pascal
Ada 83
Ada 95
Spark/Ada
Concurrent Pascal
IAL
Jovial
Simula
Eiffel
Smalltalk
JML
Java
Java5
BCPL
C
C++
C++ Standard
C#
Tcl/Tk
awk
Python
Python2.4
Perl
PHP
APL
APL2
CLOS
Lisp
Common Lisp
Scheme
Haskell
Haskell 98
Miranda
Iswim
ML
SML
OCAML
Prolog
CLP
Prolog standard

GCD Pseudocode

```
function gcd(a, b)
   while b ≠ 0
      t := b
      b := a mod t
      a := t
   return a
```

Machine Code

What was wrong with using machine code?

Machine Code

What was wrong with using machine code?

- Poor readability
- Poor modifiability
- Expression coding was tedious
- Machine deficiencies–no indexing or floating point

Plankalkül

Designed by: Konrad Zuse

Appeared in: 1943 / 1972

Features: advanced data structures:
floating point, arrays, records

never implemented

Domains: designed for engineering purposes

Meaning: “Plan Calculus”

Shortcode

Designed by: John Mauchly

Appeared in: 1949

Features:
- designed for BINAC computers
- statements represented mathematic expressions
- allowed for branching and calls to functions
- 50 times slower than machine code

Contributions:
- first higher-level language

ShortCode Example

Expressions were coded, left to right.

Example of operations:

	*		
01	–	06	ABS
02	)	07	+
03	=	08	pause
04	/	09	(
		etc	...

Example:

```
a   =   (   b   +   c   )   /   b   *  c
X3  =   (   X1  +   Y1  )   /   X1  *  Y1
X3 03 09 X1 07 Y1 02 04 X1    Y1
```

Speedcoding

Designed by:	John Backus IBM	Appeared in: Paradigm:	1953 structured

Influenced by: assembly language, machine code

Features: pseudo ops for arithmetic and math functions
conditional and unconditional branching
auto-increment registers for array access

slow!
interpreter took 310 words – 30% of the memory

Contributions: first higher-level language for an IBM computer

Fortran

Designed by:	John Backus IBM	Appeared in:	1954-57
		Paradigm:	imperative
		Extension:	.f
Influenced by:	Speedcoding		

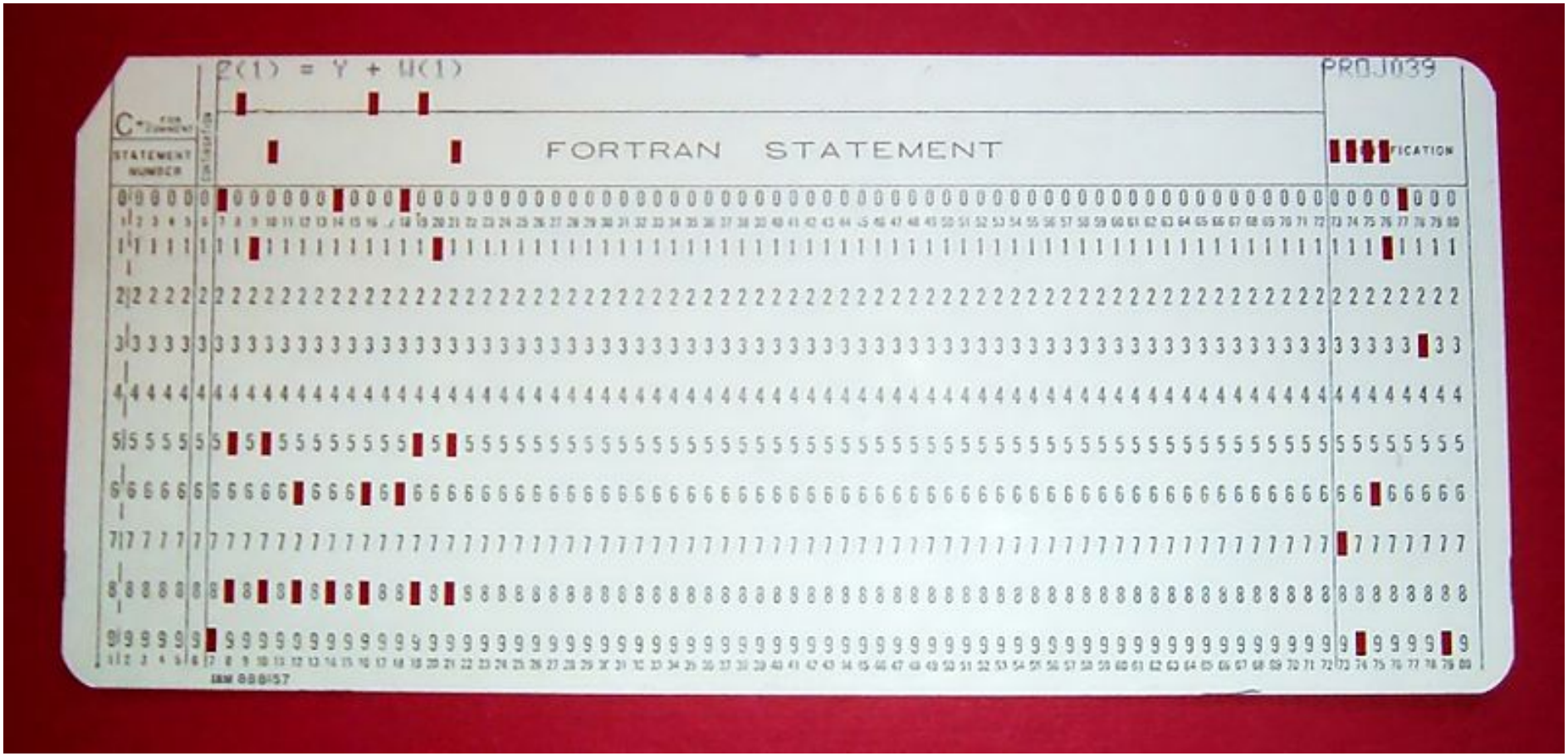
Features:
- names could have up to six characters
- formatted I/O
- user-defined subprograms
- three-way selection statement

Contributions: code was very fast, quickly became widely used

Domains: scientific and engineering applications

Acronym: IBM Mathematical **For**mula **Tran**slating System

Fortran Card



Fortran `gcd` Function

```
subroutine gcd_iter(value, u, v)
  integer, intent(out) :: value
  integer, intent(inout) :: u, v
  integer :: t

  do while( v /= 0 )
     t = u
     u = v
     v = mod(t, v)
  enddo
  value = abs(u)
end subroutine gcd_iter
```

LISP

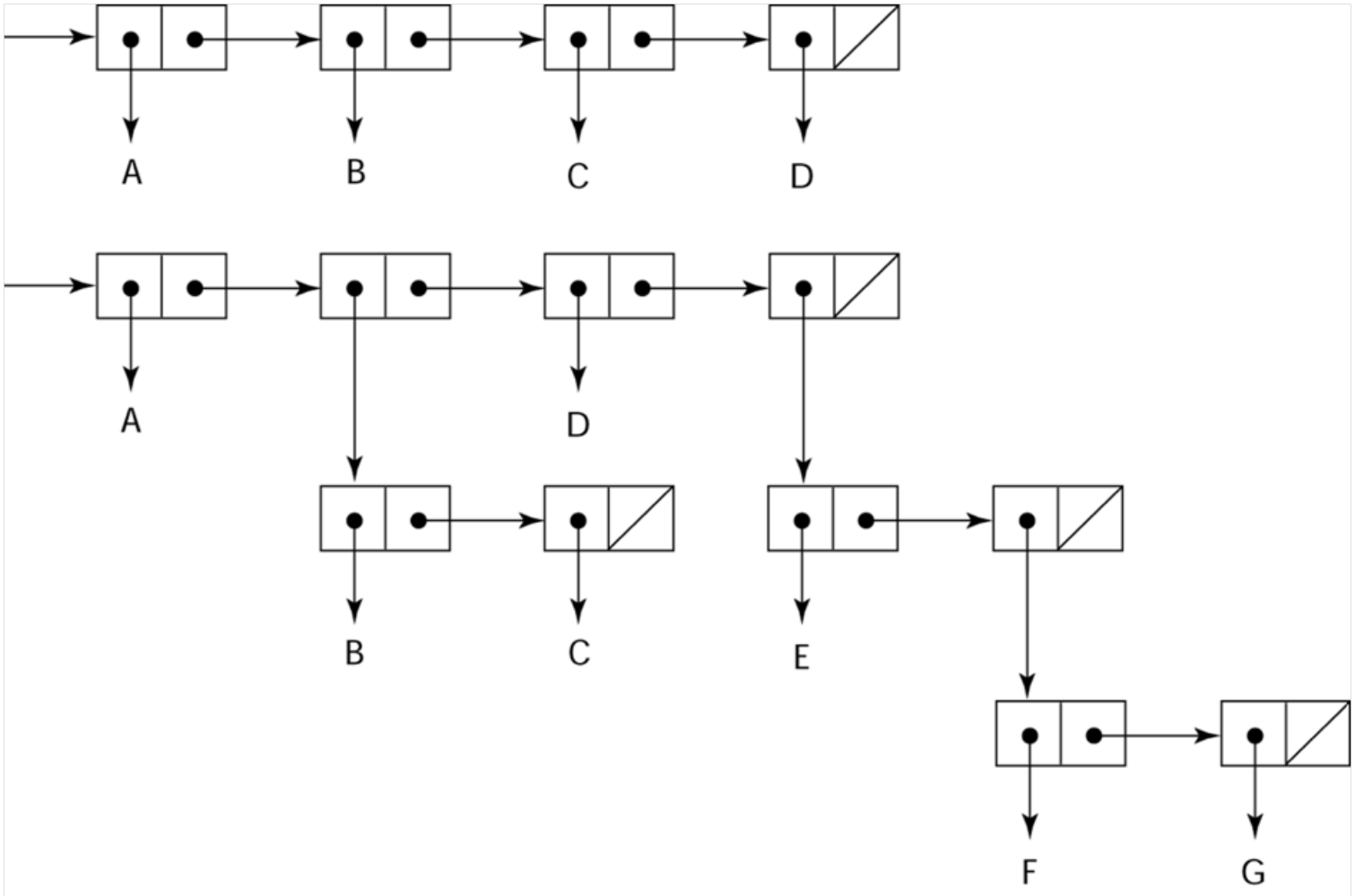
Designed by:	John McCarthy MIT	Appeared in: Paradigm: Extension:	1958 functional .lisp

Features:
- processes data in lists
- symbolic computation
- only two data types: atoms and lists
- syntax is based on lambda calculus
- control via recursion and conditional expressions

Domains: Artificial intelligence

Acronym: **LIS**t **P**rocessing language

LISP



Representation of `(A B C D)` and `(A (B C) D (E (F G)))`

LISP `gcd` Function

```
(defun gcd2 (a b)
  (do () ((zerop b) (abs a))
    (shiftf a b (mod a b))))
```

Cobol

Designed by:	Grace Hopper, *et. al*	Appeared in:	1959
		Paradigm:	imperative
		Extension:	.cbl
Influenced by:	FLOW-MATIC		

Features: data and code were completely separate, English names for arithmetic operators, long names, be easy to use, records, nested selection statements

Contributions: first macro facility in a high-level language; first language required by DoD

Domains: widely used business applications language

Acronym: **CO**mmon **B**usiness-**O**riented **L**anguage

COBOL `gcd` Function (1 / 2)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. GCD.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 A          PIC 9(10)   VALUE ZEROES.
01 B          PIC 9(10)   VALUE ZEROES.
01 TEMP       PIC 9(10)   VALUE ZEROES.

.
.
.
```

COBOL gcd Function (2 / 2)

```
PROCEDURE DIVISION.
Begin.
    DISPLAY "Enter first number, max 10 digits."
    ACCEPT A
    DISPLAY "Enter second number, max 10 digits."
    ACCEPT B
    IF A < B
      MOVE B TO TEMP
      MOVE A TO B
      MOVE TEMP TO B
    END-IF

    PERFORM UNTIL B = 0
      MOVE A TO TEMP
      MOVE B TO A
      DIVIDE TEMP BY B GIVING TEMP REMAINDER B
    END-PERFORM
    DISPLAY "The gcd is " A
    STOP RUN.
```

ALGOL 60

Designed by:	John Backus *et al.*	Appeared in:	1960
		Paradigm:	imperative

Influenced by: ALGOL 58

Features:
- concept of type was formalized
- names could be any length
- arrays could have any number of subscripts
- semicolon as a statement separator
- subprogram recursion
- stack-dynamic arrays

Contributions:
- subsequent imperative languages are based on it
- standard way to publish algorithms for over 20 years

ALGOL `gcd` Function

```
PROC gcd = (INT a, b) INT: (
  IF a = 0 THEN
    b
  ELIF b = 0 THEN
    a
  ELIF a > b  THEN
    gcd(b, a MOD b)
  ELSE
    gcd(a, b MOD a)
  FI
);
test:(
  INT a = 33, b = 77;
  printf(($x"The gcd of"g" and "g" is "gl$,a,b,gcd(a,b)));
  INT c = 49865, d = 69811;
  printf(($x"The gcd of"g" and "g" is "gl$,c,d,gcd(c,d)))
)
```

PL/I

Designed by:	IBM SHARE	Appeared in: Paradigm:	1964 imperative
Influenced by:	ALGOL, COBOL, Fortran		
Features:	designed in five months floating point, English-like syntax		
Domains:	scientific, engineering, business		
Contributions:	first unit-level concurrency, first exception handling switch-selectable recursion, first pointer data type first array cross sections		

PL/I `gcd` Function

```
GCD: procedure (a, b) returns
                          (fixed binary (31)) recursive;
   declare (a, b) fixed binary (31);

   if b = 0 then return (a);

   return (GCD (b, mod(a, b)) );

end GCD;
```

BASIC

Designed by:	John Kemeny Thomas Kurtz Dartmouth College	Appeared in:	1964
		Paradigm:	procedural
Influenced by:	ALGOL 60, FORTRAN II		
Features:	easy to learn and use		
Notes:	first widely used language with time sharing current popular dialect: Visual BASIC		
Acronym:	**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode.		

BASIC `gcd` Function

```
FUNCTION gcd(a%, b%)
    IF a > b THEN
        factor = a
    ELSE
        factor = b
    END IF
    FOR l = factor TO 1 STEP -1
        IF a MOD l = 0 AND b MOD l = 0 THEN
            gcd = l
        END IF
    NEXT l
    gcd = 1
END FUNCTION
```

APL

Designed by:	Kenneth E. Iverson IBM	Appeared in: Paradigm: Extension:	1964 functional
Influenced by:			
Features:	highly expressive dynamic typing and dynamic storage allocation		
Domains:	hardware description language		
Notes:	programs are very difficult to read still in use with minimal changes		
Acronym:	**A** **P**rogramming **L**anguage		

APL gcd Function

```
      ⌈/(^/0=A∘.|X)/A←⍳⌊/X←49865 69811
9973
```

SNOBOL

Designed by:	David J. Farber Ralph E. Griswold Bell Laboratories	Appeared in:	1964
		Paradigm:	multi-paradigm
Influenced by:			
Features:	powerful operators for string pattern matching		
Domains:	text processing tasks		
Acronym:	**S**tri**N**g **O**riented and sym**BO**lic **L**anguage		

SNOBOL `gcd` Function

```
     define('gcd(i,j)') :(gcd_end)
gcd  ?eq(i,0)  :s(freturn)
     ?eq(j,0)  :s(freturn)

loop gcd = remdr(i,j)
     gcd = ?eq(gcd,0) j :s(return)
     i = j
     j = gcd :(loop)
gcd_end

     output = gcd(1071,1029)
end
```

Simula

Designed by:	Ole-Johan Dahl Kristen Nygaard	Appeared in: Paradigm:	1967 object-oriented
Influenced by:	ALGOL 60		
Features:	classes, objects, and inheritance		
Domains:	designed for system simulation		
Contributions:	coroutines - a kind of subprogram first object-oriented programming language influenced C++		

Simula `gcd` Function

```
Begin
   Integer Procedure GCD(M, N); Integer M, N;
   Begin
      While M<>N do
         If M<N then N := N - M else M := M - N;
      GCD := M
   End of GCD;

   Integer A, B;
   OutText("Enter an integer number: ");
   OutImage; A := InInt;
   OutText("Enter an integer number: ");
   OutImage; B := InInt;
   OutText("Greatest Common Divisor of your numbers is ");
   OutInt(GCD(A,B), 4); OutImage;
End of Program;
```

Pascal

Designed by:	Niklaus Wirth	Appeared in:	1971
		Paradigm:	imperative
		Extension:	.pas
Influenced by:	ALGOL		
Features:	small, simple		
Domains:	Education		
Contributions:	From mid-1970s until the late 1990s, it was the most widely used language for teaching programming		

Pascal `gcd` Function

```
function gcd_iterative(u, v: longint): longint;
  var
    t: longint;
  begin
    while v <> 0 do
    begin
      t := u;
      u := v;
      v := t mod v;
    end;
    gcd_iterative := abs(u);
  end;
```

C

Designed by:	Dennis Richie Bell Labs	Appeared in: Paradigm: Extension:	1972 imperative .c, .h
Influenced by:	ALGOL, Assembly, PL/I, FORTRAN		
Features:	powerful set of operators poor type checking		
Domains:	designed as a systems language used in many application areas		
Contributions:	syntax influence is pervasive		

C `gcd` Function

```
int
gcd_iter(int u, int v) {
  int t;
  while (v) {
    t = u;
    u = v;
    v = t % v;
  }
  return u < 0 ? -u : u; /* abs(u) */
}
```

Prolog

Designed by:	Alain Colmerauer	Appeared in:	1972
		Paradigm:	logic
		Extension:	.pl
Influenced by:	PLANNER		
Features:	based on formal logic non-procedural		
Domains:	natural language processing, but few application areas		
Contributions:	comparatively inefficient		

Prolog `gcd` Function

```
gcd(X, 0, X):- !.
gcd(0, X, X):- !.
gcd(X, Y, D):- X > Y, !, Z is X mod Y, gcd(Y, Z, D).
gcd(X, Y, D):- Z is Y mod X, gcd(X, Z, D).
```

Smalltalk

Designed by:	Alan Kay Adele Goldberg Xerox PARC	Appeared in:	1972
		Paradigm:	object-oriented
		Extension:	.st
Influenced by:	Lisp, Simula		

Features:
- graphical user interface design
- data abstraction
- inheritance
- dynamic binding

Domains:

Contributions:
- first full implementation of an object-oriented language

Smalltalk `gcd` Function

```
|gcd_iter|

gcd_iter := [ :a :b | |u v| u := a. v := b.
    [ v > 0 ]
      whileTrue: [ |t|
          t := u copy.
          u := v copy.
          v := t rem: v
      ].
      u abs
].

(gcd_iter value: 40902 value: 24140)
  printNl.
```

Scheme

Designed by:	Guy Steele Gerald Sussman MIT	Appeared in:	1975
		Paradigm:	multi-paradigm
		Extension:	.scm
Influenced by:	ALGOL, Lisp, MDL		
Features:	extensive use of static scoping functions as first-class entities simple syntax and small size		
Domains:	Education		

Scheme `gcd` Function

```
(define (gcd a b)
  (if (= b 0)
      a
      (gcd b (modulo a b))))
```

AWK

Designed by:	Alfred Aho *et al.* Bell Labs	Appeared in:	1977
		Paradigm:	scripting
Influenced by:	C SNOBOL		

Features: extensively uses strings, hashes and reg ex's
designed to support one-liner programs
standard feature of Unix

inspired Larry Wall to write Perl

Domains: data extraction and reporting tool

Name: from its authors **A**ho, **W**einberger, and **K**ernighan

AWK `gcd` Function

```
$ awk 'func gcd(p,q)
          {return(q?gcd(q,(p%q)):p)}
          {print gcd($1,$2)}'
12 16
4
```

Matlab

Designed by:	Cleve Moler U.of New Mexico	Appeared in:	1978
		Paradigm:	multi-paradigm
		Extension:	.m, .mat
Influenced by:			
Features:	matrix manipulations plotting of functions and data		
Domains:	applied mathematics, image processing		

Matlab `gcd` Function

```
function [gcdValue] =
              greatestcommondivisor(integer1, integer2)
   gcdValue = gcd(integer1, integer2);
```

Ada

Designed by:	Jean Ichbiah MIL-STD-1815	Appeared in: Paradigm: Extension:	1980 multi-paradigm
Influenced by:	ALGOL, C++, Pascal		
Features:	generic program units packages - support for data abstraction elaborate exception handling		
Domains:	DoD		
Contributions	flexible libraries concurrency - through the tasking model		

Ada `gcd` Function

```
with Ada.Text_Io; use Ada.Text_Io;
procedure Gcd_Test is
   function Gcd (A, B : Integer) return Integer is
      M : Integer := A;
      N : Integer := B;
      T : Integer;
   begin
      while N /= 0 loop
         T := M;
         M := N;
         N := T mod N;
      end loop;
      return M;
   end Gcd;
begin
   Put_Line("GCD of 100,5 is"&Integer'Image(Gcd(100, 5)));
   Put_Line("GCD of 5,100 is"&Integer'Image(Gcd(5, 100)));
   Put_Line("GCD of 7,23 is"&Integer'Image(Gcd(7, 23)));
end Gcd_Test;
```

C++

Designed by:	Bjarne Stroustrup Bell Labs	Appeared in:	1983
		Paradigm:	multi-paradigm
		Extension:	.h, .cpp
Influenced by:	Ada, ALGOL, C, ML		
Features:	large and complex language supports both procedural and OO programming efficient compiler to native code		
Domains:	systems software, application software, embedded software		
Contributions:	rapidly grew in popularity		

C++ `gcd` Function

```
int
gcd_iter(int u, int v) {
  int t;
  while (v) {
    t = u;
    u = v;
    v = t % v;
  }
  return u < 0 ? -u : u; /* abs(u) */
}
```

Objective C

Designed by:	Brad Cox Tom Love Apple	Appeared in:	1983
		Paradigm:	object-oriented
		Extension:	.h,.m
Influenced by:	C, Smalltalk		

Features:
- C plus support for OOP based on Smalltalk
- uses Smalltalk's method calling syntax
- support for reflective features
- superset of C

Domains: used by Apple for systems programs

Perl

Designed by:	Larry Wall	Appeared in:	1987
		Paradigm:	multi-paradigm
		Extension:	.pl
Influenced by:	AWK, C++, Lisp, Pascal, Smalltalk		

Features:
- 3 distinctive namespaces, denoted in var's name
- Regular expression engine
- Variables are statically typed but implicitly declared

Domains: CGI, graphics programming, system administration, network programming, finance, bioinformatics

Backronym: **P**ractical **E**xtraction and **R**eporting **L**anguage

Perl `gcd` Function

```
sub gcd_iter($$) {
  my ($u, $v) = @_;
  while ($v) {
    ($u, $v) = ($v, $u % $v);
  }
  return abs($u);
}
```

Haskell

Designed by:	Simon Jones *et. al*	Appeared in:	1990
		Paradigm:	functional
		Extension:	.hs
Influenced by:	Lisp, ML Scheme		

Features: primary control construct is the function
non-strict semantics and strong static typing

Haskell `gcd` Function

```
gcd :: (Integral a) => a -> a -> a
gcd 0 0 =  error "Prelude.gcd: gcd 0 0 is undefined"
gcd x y =  gcd' (abs x) (abs y) where
  gcd' a 0  =  a
  gcd' a b  =  gcd' b (a `rem` b)
```

Python

Designed by:	Guido van Rossum	Appeared in:	1991
		Paradigm:	multi-paradigm
		Extension:	.py
Influenced by:	ALGOL, C, C++, Haskell, Java, Lisp, Perl		
Features:	OO interpreted scripting language type checked but dynamically typed supports lists, tuples, and hashes		
Domains:	CGI programming, form processing		

Python `gcd` Function

```
def gcd_iter(u, v):
    while v:
        u, v = v, u % v
    return abs(u)
```

Lua

Designed by:	R. Ierusalimschy W. Celes L.H. de Figueiredo	Appeared in:	1993
		Paradigm:	multi-paradigm
Influenced by:	C++, Modula, Scheme		

Features:
- OO interpreted scripting language
- type checked but dynamically typed
- single data structure – table
- easily extendable

Domains:
- CGI programming, form processing

Means:
- “moon” in Portuguese

Lua `gcd` Function

```
function gcd(a,b)
    if b ~= 0 then
        return gcd(b, a % b)
    else
        return math.abs(a)
    end
end

function demo(a,b)
    print("GCD of "..a.." and "..b.." is "..gcd(a, b))
end
```

JavaScript

Designed by:	Brendan Eich Netscape	Appeared in: Paradigm: Extension:	1994 multi-paradigm .js
Influenced by:	C, Java, Perl, Python, Scheme		
Features:	client-side HTML-embedded scripting language purely interpreted		
Domains:	dynamic HTML documents		

JavaScript `gcd` Function

```
function gcd(a,b) {
    if (a < 0) a = -a;
    if (b < 0) b = -b;
    if (b > a) {var temp = a; a = b; b = temp;}
    while (true) {
        a %= b;
        if (a == 0) return b;
        b %= a;
        if (b == 0) return a;
    }
}
```

PHP

Designed by:	Rasmus Lerdorf	Appeared in:	1995
		Paradigm:	imperative, OO
		Extension:	.php
Influenced by:	C, C++, Java, Perl		
Features:	server-side scripting language purely interpreted		
Domains:	form processing and database access		
Acronym:	**P**HP: **H**ypertext **P**reprocessor		

PHP `gcd` Function

```
function gcdIter($n, $m) {
    while(true) {
        if($n == $m) {
            return $m;
        }
        if($n > $m) {
            $n -= $m;
        } else {
            $m -= $n;
        }
    }
}
```

Ruby

Designed by:	Yukihiro Matsumoto	Appeared in:	1995
		Paradigm:	multi-paradigm
		Extension:	.rb
Influenced by:	Ada, C++, Lisp, Perl, Python, Smalltalk		

Features:

- pure object-oriented scripting language
- purely interpreted
- operators are implemented as methods

Ruby `gcd` Function

```ruby
def gcd(u, v)
  u, v = u.abs, v.abs
  while v > 0
    u, v = v, u % v
  end
  u
end
```

Java

Designed by:	James Gosling Sun Microsystems	Appeared in:	1995
		Paradigm:	multi-paradigm
		Extension:	.java, .class
Influenced by:	Ada, C++, C#, Pascal, Smalltalk		

Features:

- supports only OOP
- references, but not pointers
- support for applets
- supports concurrency
- Java Virtual Machine concept
- libraries for applets, GUIs, database access
- widely used for Web programming

Java `gcd` Function

```
public static long gcd(long a, long b){
    long factor= Math.max(a, b);
    for(long loop= factor;loop > 1;loop--){
        if(a % loop == 0 && b % loop == 0){
            return loop;
        }
    }
    return 1;
}
```

OCaml

Designed by:	Xavier Leroy *et al.*	Appeared in:	1996
		Paradigm:	functional, OO
Influenced by:	Standard ML		
Features:	large standard library robust object-oriented programming constructs static type system		
Name:	Objective Caml		

Caml `gcd` Function

```
let rec gcd a b =
  if       a = 0 then b
  else if b = 0 then a
  else if a > b then gcd b (a mod b)
  else               gcd a (b mod a)
```

C#

Designed by:	Microsoft	Appeared in:	2000
		Paradigm:	multi-paradigm
		Extension:	.cs
Influenced by:	C++, Java, Pascal		
Features:	includes pointers, delegates, properties, enumeration types, limited kind of dynamic typing, anonymous types		
Domains:	.NET		
Contributions:	is evolving rapidly		

C# `gcd` Function

```
private static int gcd(int a, int b)
{
    int t;
    // Ensure B > A
    if (a > b)
    {
        t = b;
        b = a;
        a = t;
    }
    // Find
    while (b != 0)
    {
        t = a % b;
        a = b;
        b = t;
    }
    return a;
}
```

Go

Designed by:	Robert Griesemer *et al.* Google	Appeared in: Paradigm: Extension:	2009 imperative .go
Influenced by:	C, Modula, Pascal, Python		

Features:

loosely based on C, but also quite different
does not support traditional OOP
goroutines, small lightweight threads
visibility according to capitalization
efficient, latency-free garbage collection
line-ending semicolons are optional
designed for exceptionally fast compiling times

Go `gcd` Function

```
package main

import "fmt"

func gcd(x, y int) int {
    for y != 0 {
        x, y = y, x%y
    }
    return x
}

func main() {
    fmt.Println(gcd(33, 77))
    fmt.Println(gcd(49865, 69811))
}
```

Rust

Designed by:	Graydon Hoare *et al.* Mozilla Research	Appeared in:	2012
		Paradigm:	multi-paradigm
		Extension:	.rs
Influenced by:			

Features:

- designed for large client and server programs
- syntax similar to subset of C and C++
- memory safe (no null or dangling pointers
- type system supports 'traits', inspired by Haskell
- features type inference
- supports concurrency
- performance of safe code is slower than C++
- sponsored by Mozilla and Samsung
- open community project

Rust `gcd` Function

```
fn gcd(mut m: int, mut n: int) -> int {
    while m != 0 {
        let temp = m;
        m = n % temp;
        n = temp;
    }
    n.abs()
}

fn gcd(m: int, n: int) -> int {
    if m == 0
        { n.abs() }
    else
        { gcd(n % m, m) }
}
```