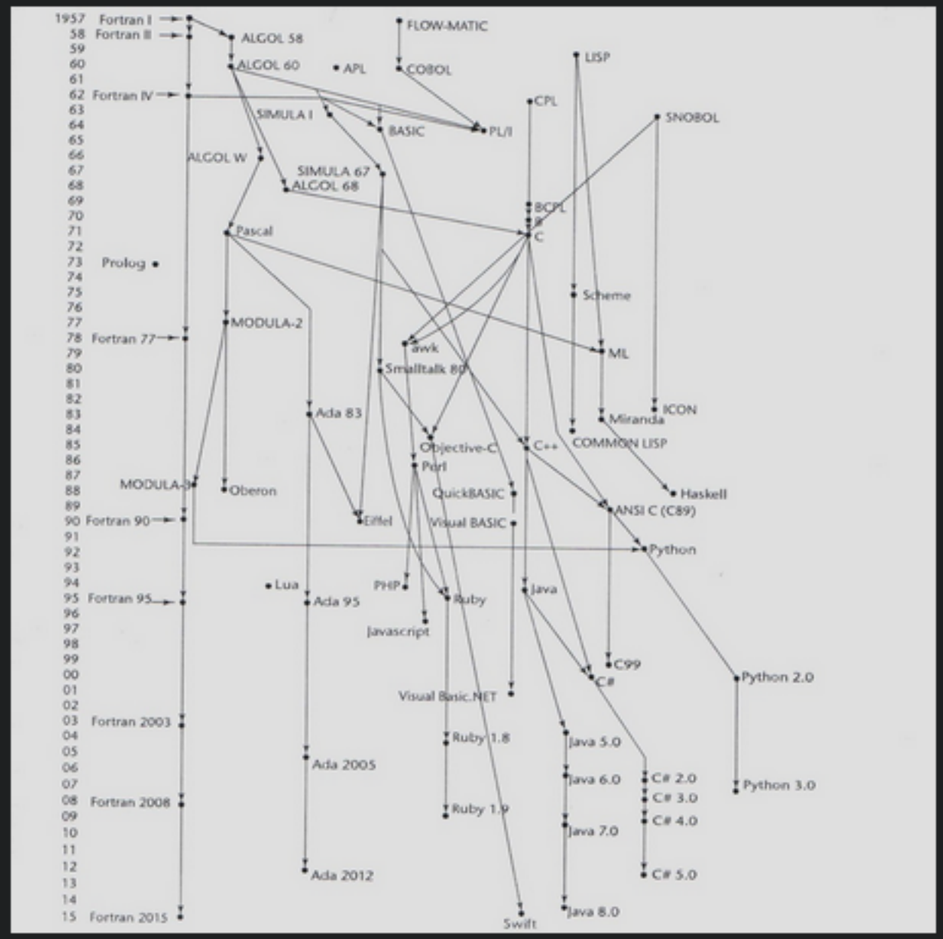
Linguaggi di Programmazione ad Alto Livello

- principali linguaggi imperativi (e a oggetti)
- principali linguaggi funzionali
- principali linguaggi logici
- principali linguaggi di scripting
- principali linguaggi di markup



Evoluzione dei Linguaggi di Programmazione

Abbiamo già parlato di...

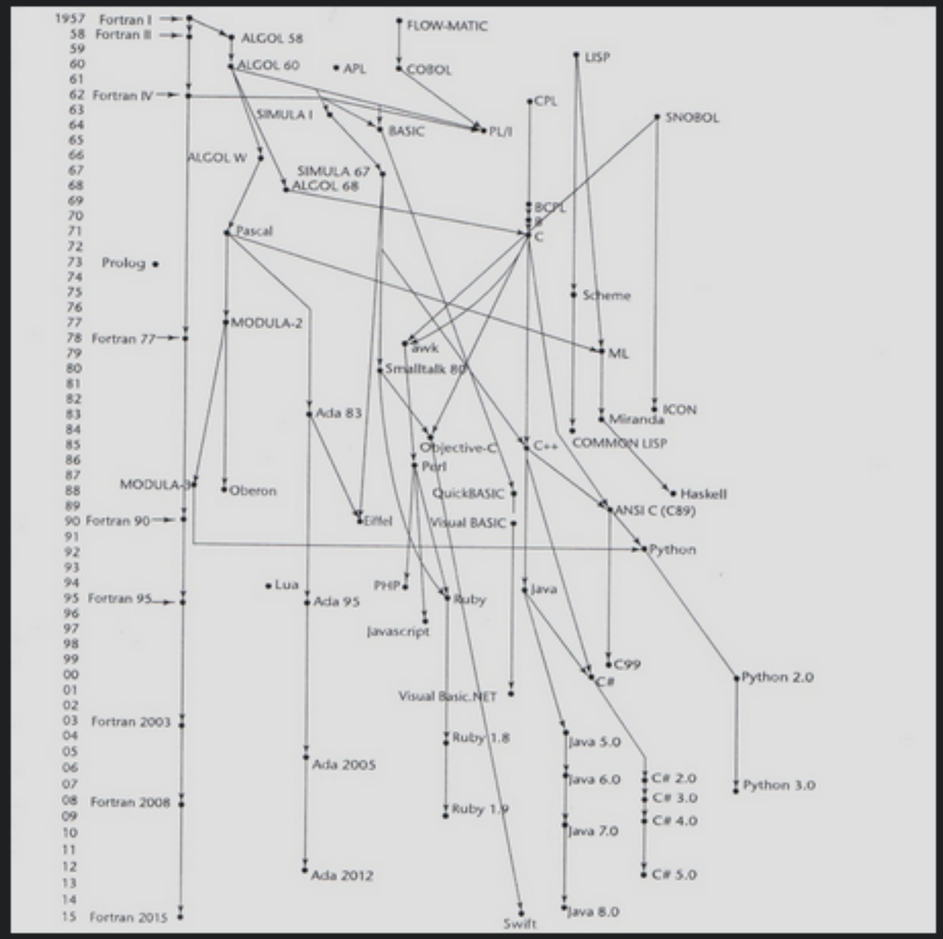
- **Plankalkül di Zuse**
- Progettato nel 1945, ma non pubblicato fino al 1972
- Mai implementato
- Strutture dati avanzate – virgola mobile, array, record
- Invarianti
- Cosa c'era di sbagliato nell'usare il codice macchina? – Scarsa leggibilità – Scarsa modificabilità – La codifica delle espressioni era tediosa – Carenze della macchina -- nessuna indicizzazione o virgola mobile

Pseudocodici

- **Short Code** sviluppato da Mauchly nel 1949 per i computer BINAC
  - Le espressioni venivano codificate da sinistra a destra, usando codici menominici di due caratteri invece delle cifre binarie
  - Esempio di operazioni: **01** –; **06** valore assoluto; **1n** elevamento alla (n+2); **02**); **07** +; **2n** radice (n+2); **03** =; **4n** if <= n; **04** /; **09** (; **58** stampa e tabulazione
- **Speedcoding** sviluppato da Backus nel 1954 per IBM 701
  - Pseudo-operazioni per funzioni aritmetiche e matematiche
  - Ramificazione condizionale e incondizionale; Registri auto-incrementanti per l'accesso agli array
  - Lento! Solo 700 parole rimaste per il programma utente
- Nel **Sistema di Compilazione UNIVAC**, sviluppato da un team guidato da Grace Hopper, lo pseudocodice veniva espanso in codice macchina;

Linguaggi di Programmazione ad Alto Livello

- principali linguaggi imperativi (e a oggetti)
- principali linguaggi funzionali
- principali linguaggi logici
- principali linguaggi di scripting
- principali linguaggi di markup



Programmazione Imperativa

IBM 704 e Fortran

- Fortran 0: 1954 - non implementato
- Fortran I: 1957 – Progettato per il nuovo IBM 704, che aveva registri indice e hardware per virgola mobile
  - Questo portò all'idea dei linguaggi di programmazione compilati, perché non c'era posto dove nascondere il costo dell'interpretazione (nessun software per virgola mobile)
- Ambiente di sviluppo
  - I computer erano piccoli e inaffidabili
  - Le applicazioni erano scientifiche
  - Nessuna metodologia o strumento di programmazione
  - L'efficienza della macchina era la preoccupazione più importante

Processo di Progettazione del Fortran

- Impatto dell'ambiente di sviluppo richiesto sulla progettazione del Fortran I, prima versione implementata del Fortran
- Nessun bisogno di memorizzazione dinamica
- Necessità di buona gestione degli array e cicli di conteggio
- Nessuna gestione di stringhe, aritmetica decimale o input/output potente (per software aziendale)
- I nomi potevano avere fino a sei caratteri; Ciclo di conteggio post-test (**DO etichetta dell'ultima istruzione var = primo, ultimo)**; I/O formattato; Sottoprogrammi definiti dall'utente; Istruzione di selezione a tre vie (**IF** aritmetico N-, N0, N+); Nessuna dichiarazione di tipo di dato

Esempi di Codice

```
 IF (espressione) etichetta_negativa, etichetta_zero, etichetta_positiva

 X = 15.0

Y = 10.0       C     Per testare se X > Y, si calcola X-Y

 IF (X - Y) 10, 20, 30

10    PRINT 101

      GO TO 40

20    PRINT 102     ! X = Y (X-Y zero)

      GO TO 40

30    PRINT 103

40    CONTINUE
```

Esempi di Codice

- **Regola implicita I-N**: Le variabili che iniziavano con le lettere I, J, K, L, M, N erano automaticamente **INTEGER**, tutte le altre erano **REAL** (numeri in virgola mobile)
- **Nessuna dichiarazione**: Non esisteva una sezione di dichiarazione delle variabili. Bastava usarle nel codice e il compilatore applicava automaticamente la regola I-N. Questo approccio rendeva la programmazione più immediata per i matematici e ingegneri dell'epoca

```
I = 5 ! INTEGER (inizia con I)

N = 10 ! INTEGER (inizia con N)

X = 3.14 ! REAL (inizia con X)

TEMP = 25.0 ! REAL (inizia con T)

DO 10 J = 1, N ! J è INTEGER automaticamente

  X = X + J ! X è REAL, J è INTEGER
```

Panoramica del Fortran I

- Prima versione implementata del FORTRAN
- Nessuna compilazione separata per le subroutines
- il codice doveva essere scritto secondo un formato fisso (**schede perforate**), es. etichette nei primi 5 caratteri della riga, istruzioni non prima del settimo carattere
- tipo e dimensioni delle variabili dichiarati in modo statico a compile-time; di conseguenza, no ricorsione
- Compilatore rilasciato nell'aprile 1957, dopo 18 anni-uomo di lavoro
- I programmi più grandi di 400 righe raramente compilavano correttamente, principalmente a causa della scarsa affidabilità del 704
- Il codice era molto veloce
- Divenne, rapidamente, ampiamente utilizzato

Fortran II e Fortran IV, Fortran 77 e Fortran 90

- Distribuito nel 1958 – Compilazione indipendente per le subroutines – Corretti i bug
- **Evoluzione durante il 1960-62** – Dichiarazioni di tipo esplicite – Istruzione di selezione logica – I nomi dei sottoprogrammi potevano essere parametri – Standard ANSI nel 1966
- Divenne il nuovo standard nel 1978 – Gestione di stringhe di caratteri – Istruzione di controllo ciclo logico – Istruzione **IF-THEN-ELSE**
- Modifiche più significative dal Fortran 77 – Moduli – Array dinamici e Puntatori gestiti dinamicamente (Allocatable)– Ricorsione – Istruzione **CASE** – Controllo del tipo dei parametri

Versioni più recenti di Fortran e Valutazione

- Fortran 95 – aggiunte relativamente minori, più alcune eliminazioni
- Fortran 2003 – supporto per OOP, puntatori a procedure, interoperabilità con C
- Fortran 2008 – blocchi per ambiti locali, co-array, Do Concurrent
- Compilatori altamente ottimizzanti (tutte le versioni prima del 90) – Tipi e memorizzazione di tutte le variabili sono fissati prima dell'esecuzione
- Ha cambiato per sempre il modo di usare i computer

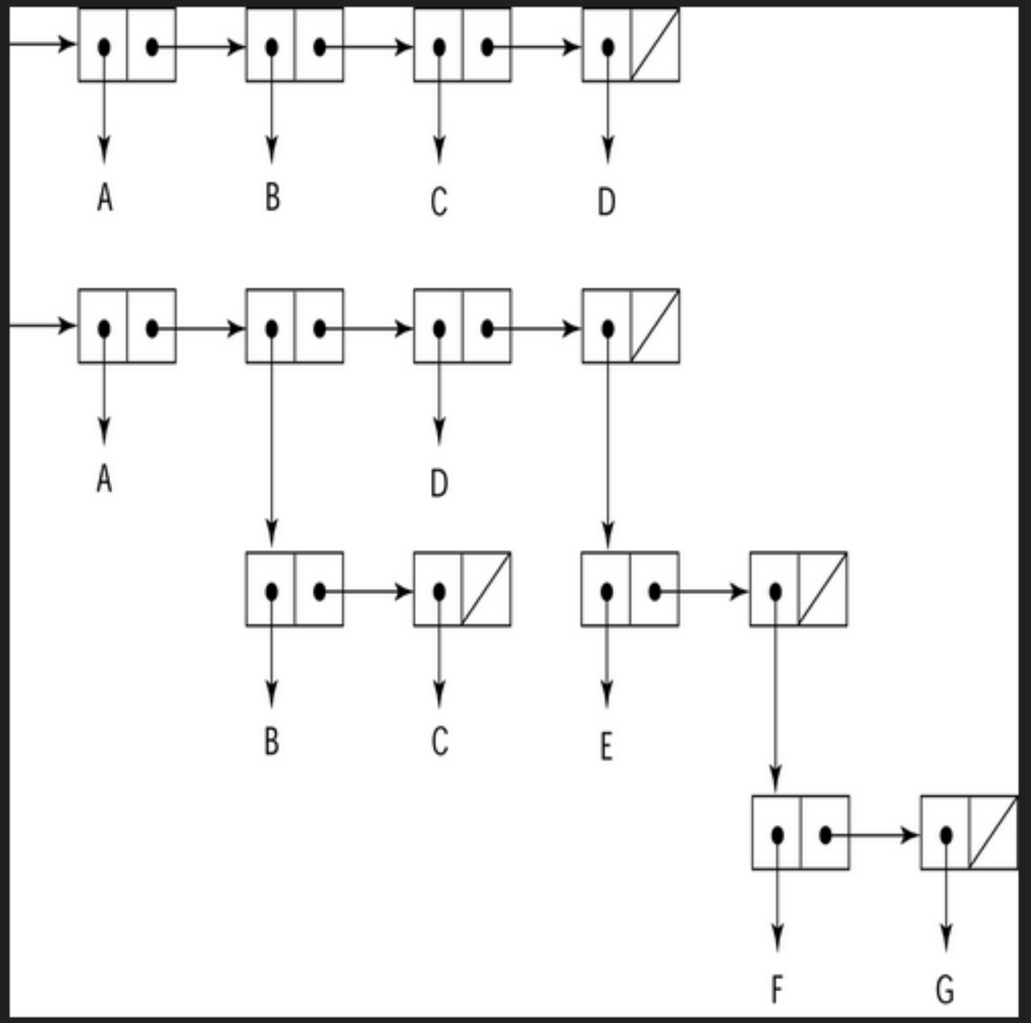
Programmazione Funzionale

LISP: LISt Processing

- Linguaggio di elaborazione di LISte, progettato al MIT da McCarthy
- La ricerca in IA necessitava di un linguaggio per elaborare dati simbolici – Elaborare dati in liste (piuttosto che array) – Calcolo simbolico (piuttosto che numerico)
- Solo due tipi di dato: atomi (valori numerici o letterali) e liste (i cui elementi possono essere atomi o liste)
- il codice e i dati hanno lo stesso formato: (f a b c d) può essere una funzione o un dato a seconda di come viene interpretato
- La sintassi è basata sul *lambda calcolo:*
  - **trasparenza referenziale:** svincolarsi dall'idea di stato tipico dell'architettura di Von Neumann
  - il calcolo è dato dall'applicazione di funzione agli argomenti
  - niente assegnamento, nè variabili, nè cicli, solo ricorsione

Strutture Dati Fondamentali

- una lista piatta di **atomi** (a b c d)
- una lista composta da altre liste
- ogni lista è costruita a partire da **cons-cell**
- una cons-cell è una coppia di puntatori (car, cdr)



Valutazione di Lisp

- Linguaggio pionieristico e di riferimento per la programmazione funzionale
- Nessun bisogno di variabili o assegnamento (**trasparenza referenziale**)
- Controllo tramite ricorsione ed espressioni condizionali
- Ancora il linguaggio dominante per l'IA
- Common Lisp e Scheme sono dialetti contemporanei di Lisp
- ML, Haskell e Julia sono a loro volta linguaggi di programmazione funzionale, ma usano sintassi molto diverse

I Dialetti di LISP

- **SCHEME**: Sviluppato al MIT a metà degli anni '70
  - Uso estensivo dello scoping statico
  - Funzioni come entità di prima classe
  - Sintassi semplice e dimensioni ridotte lo rendono ideale per applicazioni educative
- **COMMON LISP**: Uno sforzo per combinare caratteristiche di diversi dialetti di Lisp in un singolo linguaggio
  - Grande, complesso, usato nell'industria per alcune grandi applicazioni
  - Lo utilizzeremo durante il nostro corso

Altri linguaggi Funzionali: ML e Haskell

- ML: strutture dati omogenee, il tipo di ogni variabile ed espressione può essere determinato a compile-time (strong, static typing), i tipi vengono inferiti automaticamente dal contesto di un' espressione (type inference), include costrutti imperativi
- Haskell: il tipo di ogni variabile ed espressione può essere determinato a compile-time (strong, static typing), i tipi vengono inferiti automaticamente dal contesto di un' espressione (type inference), funzioni definite per **currificazione** e **valutazione lazy** delle stesse

Torniamo alla Programmazione Imperativa

ALGOL 60

- Ambiente di sviluppo altamente specializzato:
  - FORTRAN era (appena) arrivato per IBM 70x, molti altri linguaggi erano in sviluppo, tutti per macchine specifiche; nessun linguaggio portabile, tutti erano dipendenti dalla macchina
  - Nessun linguaggio universale per comunicare algoritmi: petizione alla ACM (americana) per formare un comitato e creare un linguaggio universale, GAMM (associazione tedesca di matematica applicata) già al lavoro
- ALGOL 60 fu il risultato degli sforzi per progettare un linguaggio universale:
  - più vicino possibile alla notazione matematica, e leggibile con pochi altri commenti
  - che potesse essere utilizzato per descrivere processi di calcolo, anche in pubblicazioni scientifiche
  - traducibile meccanicamente in linguaggio macchina

Processo di Progettazione Iniziale: ALGOL 58

- ACM e GAMM si incontrarono per quattro giorni per la progettazione (27 maggio - 1 giugno 1958)
- Obiettivi del linguaggio: vicino alla notazione matematica, descrivere algoritmi, traducibile in codice macchina
- Molte caratteristiche che conosciamo: Il concetto di **tipo fu formalizzato,** i nomi potevano essere di qualsiasi lunghezza, gli array potevano avere qualsiasi numero di indici, gli indici erano posti tra **parentesi quadre**, **blocchi** di istruzioni (**begin ... end**), punto e virgola come separatore di istruzioni, **if** aveva una clausola **else-if, L'operatore di assegnamento era := (tipico dei linguaggi ALGOL-like)**
- I parametri erano separati per modalità (in & out)
- Nessun I/O - "lo renderebbe dipendente dalla macchina"

Dal ALGOL 58 ad ALGOL 60

- ALGOL 58 modificato in un incontro di 6 giorni a Parigi
- Nuove caratteristiche: struttura a blocchi (ambito locale) , due metodi di passaggio parametri, ricorsione dei sottoprogrammi, array stack-dinamici , ancora nessun I/O e nessuna gestione stringhe
- La sintassi viene definita formalmente tramite BNF da Naur, per la prima volta
- Nonostante i propositi iniziali, non fu mai veramente usato e rimase soprattutto un linguaggio utile per scrivere codice negli articoli (mancanza di supporto da parte di IBM e rafforzamento di FORTRAN)

```
comment ALGOL 60 Example Program
 Input: ...
 Output: ...
begin
    integer array intlist [1:99];
    integer listlen, counter, sum;
    real average;

    sum := 0;
    comment No built-in I/O facilities!
    readint(listlen);
    if (listlen > 0) ^ (listlen < 100) then
        begin
            for counter := 1 step 1 until listlen do
                begin
                    readint(intlist[counter]);
                    sum := sum + intlist[counter]
                end
            average := sum / counter;
            printstring(‘‘The average is’’);

            printreal(average)
        end
    else
        printstring(‘‘error’’);
end
```

Programmazione Logica

Un Linguaggio solo al comando: PROLOG

- Sviluppato negli anni '70 da Comerauer e Roussel (Università di Aix-Marseille), supportati da Kowalski (Università di Edimburgo)
- Basato sulla logica formale
- Non procedurale, ma dichiarativo
- Idea di base: un programma è un insieme di fatti e regole che servono a dimostrare un teorema (VERO o FALSO)
- fatti (es. madre(anna, silvia)), regole (es. nonna(X,Y) :- madre(X,Z), madre(Z,Y))
- Relativamente inefficiente
- Poche aree applicative

Torniamo alla Programmazione Imperativa

COBOL: COmmon Business-Oriented Language

- A fine anni '50, UNIVAC stava iniziando a usare FLOW-MATIC, USAF stava iniziando a usare AIMACO – IBM stava sviluppando COMTRAN
- Su mandato del Dipartimento di Difesa U.S.A., il comitato CODASYL (Committee on Data Systems Languages) che sviluppò COBOL attinse da tutti questi linguaggi
- Obiettivi di progettazione: deve sembrare inglese semplice; deve essere facile da usare, anche se significa essere meno potente; deve ampliare la base di utenti di computer; non deve essere influenzato dai problemi attuali dei compilatori
- Problemi di progettazione: espressioni aritmetiche? indici? Conflitti tra produttori

Valutazione di COBOL

- Contributi: prima struttura macro (DEFINE) in un linguaggio di alto livello; strutture dati gerarchiche (record), Istruzioni di selezione nidificate, nomi lunghi (fino a 30 caratteri), con hypehs, dichiarazione esplicita del numero di bit per parte intera e decimale per ogni variabile
- Primo linguaggio richiesto dal DoD – sarebbe fallito senza il DoD
- Ancora oggi uno dei linguaggi più usato per applicazioni aziendali
- Debole nella parte procedurale:
  - niente funzioni (solo procedure)
  - le prime versioni non permettevano passaggio di parametri

Beginner's All-purpose Symbolic Instruction Code: BASIC

- Progettato da Kemeny & Kurtz a Dartmouth
- Obiettivi di Progettazione: facile da imparare e usare per studenti non scientifici, Deve essere "piacevole e amichevole", input outup rapido per lo svolgimento degli esercizi, accesso gratuito e privato, il **tempo dell'utente** è più importante del tempo del computer
- solo 14 tipi diversi di comandi
- un singolo tipo di dati (numbers = float)
- Dialetto popolare attuale: Visual Basic, storicamente legato ad ambienti visuali e prodotti Microsoft
- Primo linguaggio ampiamente usato con **time sharing**

Due Linguaggi Dinamici Precoci: APL e SNOBOL

- Caratterizzati da tipizzazione dinamica e allocazione dinamica della memoria
- Le variabili non sono tipizzate – Una variabile acquisisce un tipo quando le viene assegnato un valore
- La memoria è allocata a una variabile quando le viene assegnato un valore
- **APL**: Progettato come linguaggio di descrizione hardware alla IBM da Ken Iverson intorno al 1960; altamente espressivo (molti operatori, sia per scalari che array di varie dimensioni); i programmi sono molto difficili da leggere; ancora in uso; modifiche minime
- **SNOBOL**: Progettato come linguaggio di manipolazione stringhe ai Bell Labs da Farber, Griswold e Polensky nel 1964
- Operatori potenti per il pattern matching delle stringhe
- Più lento dei linguaggi alternativi (e quindi non più usato per scrivere editor); ancora usato per certi compiti di elaborazione testo

L'Inizio dell'Astrazione dei Dati: SIMULA 67

- Progettato principalmente per la simulazione di sistemi in Norvegia da Nygaard e Dahl
- Basato su ALGOL 60 e SIMULA I
- Contributi Principali:
  - Coroutine - un tipo di sottoprogramma
  - Classi, oggetti ed ereditarietà

Progettazione Ortogonale: ALGOL 68

- Derivato dallo sviluppo continuato di ALGOL 60 ma non un superset di quel linguaggio
- Fonte di diverse nuove idee (anche se il linguaggio stesso non raggiunse mai uso diffuso)
- La progettazione è basata sul concetto di ortogonalità: pochi concetti base, ancor meno meccanismi di combinazione
- Contributi: strutture dati definite dall'utente, tipi reference (puntatori); array dinamici (chiamati flex arrays)
- Commenti: meno usato di ALGOL 60; ebbe forte influenza sui linguaggi successivi, specialmente Pascal, C e Ada

Pascal - 1971

- Sviluppato da Wirth (ex membro del comitato ALGOL 68)
- Progettato per insegnare programmazione strutturata
- Piccolo, semplice, niente di veramente nuovo
- L'impatto maggiore fu sull'insegnamento della programmazione; da metà anni '70 fino fine anni '90, fu il linguaggio più usato per insegnare programmazione

C - 1972

- Progettato per programmazione di sistema (ai Bell Labs da Dennis Ritchie)
- Evoluto principalmente da BCLP e B, ma anche ALGOL 68
- Set potente di operatori, ma scarso controllo dei tipi
- Inizialmente diffuso attraverso UNIX
- Anche se progettato come linguaggio di sistema, è stato usato in molte aree applicative

Il Più Grande Sforzo di Progettazione della Storia: Ada

- Enorme sforzo di progettazione, coinvolgendo centinaia di persone, molto denaro e circa otto anni
- Sequenza di requisiti (1975-1978) – (Strawman, Woodman, Tinman, Ironman, Steelman)
- Chiamato Ada da Augusta Ada Byron, la prima programmatrice
- Contributi: Pacchetti, supporto per astrazione dati, Gestione eccezioni elaborata, unità di programma generiche, concorrenza attraverso il modello di tasking
- Commenti: Progettazione competitiva, includeva tutto quello che allora si sapeva su ingegneria software e progettazione linguaggi; i primi compilatori erano molto difficili; il primo compilatore veramente utilizzabile arrivò quasi cinque anni dopo il completamento della progettazione del linguaggio
- Lo sviluppo continuò negli anni '90 e 2000 con Ada 95 e Ada 2005, versioni OO di scarso successo perchè in competizione con C++

Programmazione Orientata Agli Oggetti

Smalltalk

- Sviluppato a Xerox PARC, inizialmente da Alan Kay, poi da Adele Goldberg
- Prima implementazione completa di un linguaggio orientato agli oggetti (astrazione dati, ereditarietà e binding dinamico)
- Pioniere della progettazione dell'interfaccia utente grafica
- Promosse OOP

Combinando Programmazione Imperativa e Orientata agli Oggetti...C++

- Sviluppato ai Bell Labs da Stroustrup nel 1980
- Evoluto da C e SIMULA 67
- Strutture per programmazione orientata agli oggetti, prese parzialmente da SIMULA 67
- Un linguaggio grande e complesso, perché supporta sia programmazione procedurale che OO
- Crebbe rapidamente in popolarità, insieme a OOP
- Standard ANSI approvato nel novembre 1997
- Negli anni, varie versioni sviluppate a seconda delle piattaforme

Un Linguaggio OO Basato su Imperativo: Java

- Sviluppato da Sun nei primi anni '90: C e C++ non erano soddisfacenti per dispositivi elettronici embedded
- Basato su C++, significativamente semplificato (non include **struct, union, enum**, aritmetica puntatori e metà delle coercizioni di assegnazione di C++); supporta *solo* OOP; ha riferimenti, ma non puntatori; include supporto per applet e una forma di concorrenza (i Thread)
- Eliminò molte caratteristiche non sicure di C++
- Supporta concorrenza
- Librerie per applet, GUI, accesso database
- Portabile: concetto Java Virtual Machine, compilatori JIT
- Usato per programmazione Web; l'uso è aumentato più velocemente di qualsiasi linguaggio precedente

Linguaggi di Scripting per il Web

- **Perl:** Progettato da Larry Wall (1987), le variabili sono tipizzate staticamente ma dichiarate implicitamente. Tre namespace distintivi, denotati dal primo carattere del nome di una variabile – Potente, ma alquanto pericoloso – Ottenne uso diffuso per programmazione CGI sul Web
- **JavaScript:** iniziò a Netscape, ma poi divenne joint venture di Netscape e Sun Microsystems; un linguaggio di scripting lato client HTML - embedded, spesso usato per creare documenti HTML dinamici; puramente interpretato, correlato a Java solo attraverso sintassi simile
- **PHP:** PHP: Hypertext Preprocessor, progettato da Rasmus Lerdorf; un linguaggio di scripting lato server HTML-embedded, spesso usato per elaborazione form e accesso database attraverso il Web; puramente interpretato
- **Python: u**n linguaggio di scripting interpretato OO; controllo tipo ma tipizzazione dinamica; usato per elaborazione form; inferenza di tipo; supporta liste, tuple e hash
- **Ruby: p**rogettato in Giappone da Yukihiro Matsumoto (alias "Matz"); iniziò come sostituto per Perl e Python; un linguaggio di scripting puramente orientato agli oggetti. Tutti i dati sono oggetti, puramente interpretato

Linguaggi Ibridi Markup/Programmazione

- **XSLT** – eXtensible Markup Language (XML): un meta-linguaggio di markup – eXtensible Stylesheet Language Transformation (XSLT) trasforma documenti XML per la visualizzazione – Costrutti di programmazione (es. cicli)
- **JSP** – Java Server Pages: una raccolta di tecnologie per supportare documenti Web dinamici – JSTL, una libreria JSP, include costrutti di programmazione sotto forma di elementi HTML