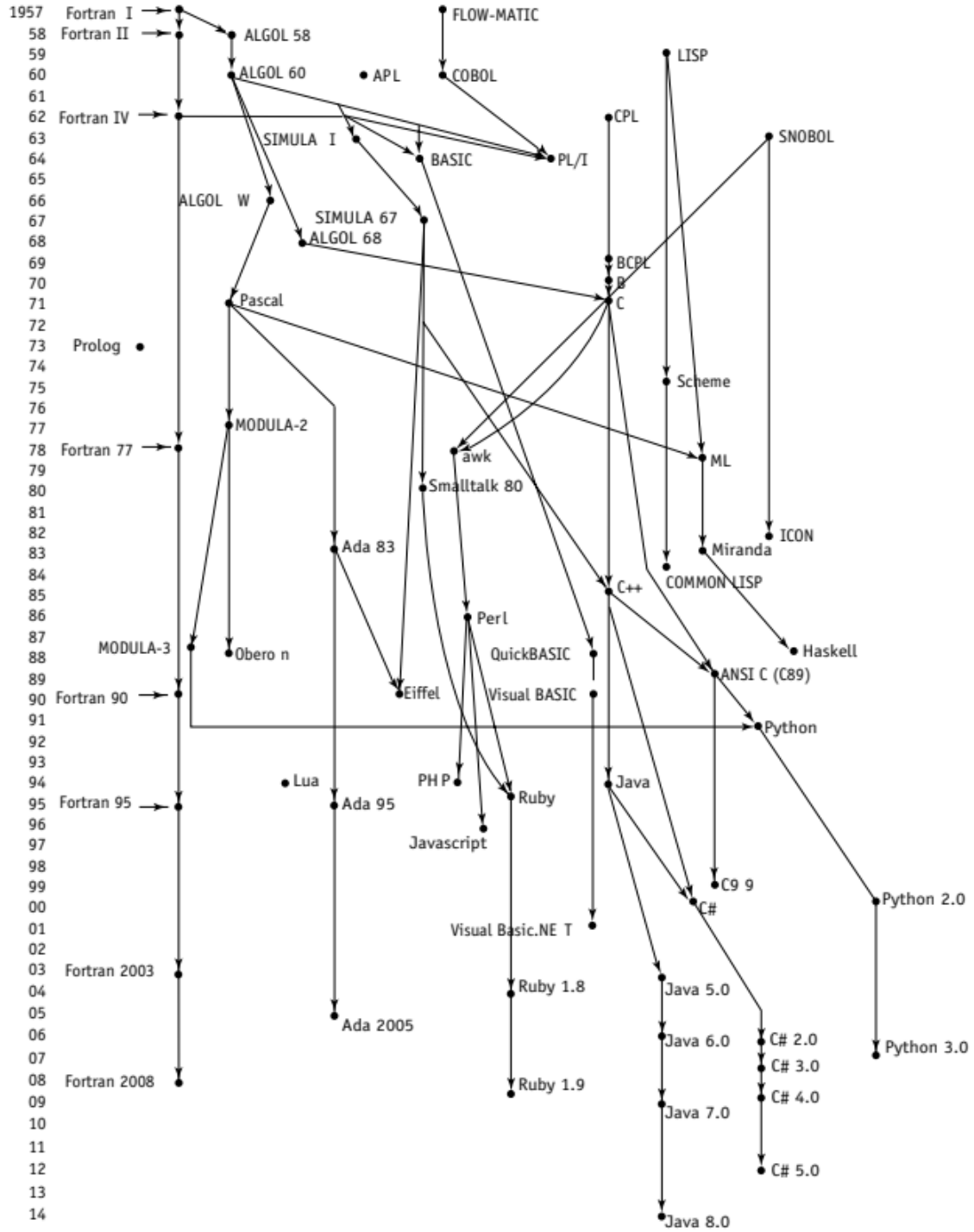
Paradigmas de Linguagens de Programação

Histórico das Linguagens de Programação

Genealogia das principais linguagens de programação de alto nível.

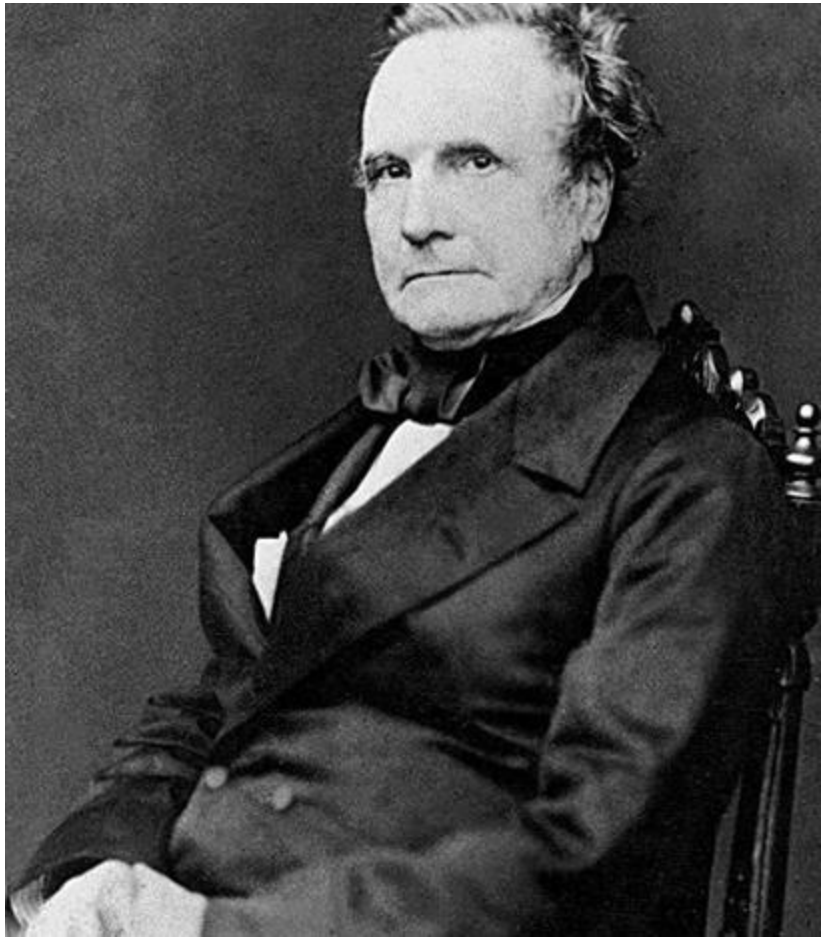


Ada Lovelace

- Augusta Ada, Condessa de Lovelace
- Filha de Lord Byron,um grande poeta Inglês.
- Início dos anos 1800: educação feminina era restrita
- Ada tomou aulas particulares de matemática.
- Casou-se aos 19, tornando-se Lady Lovelace

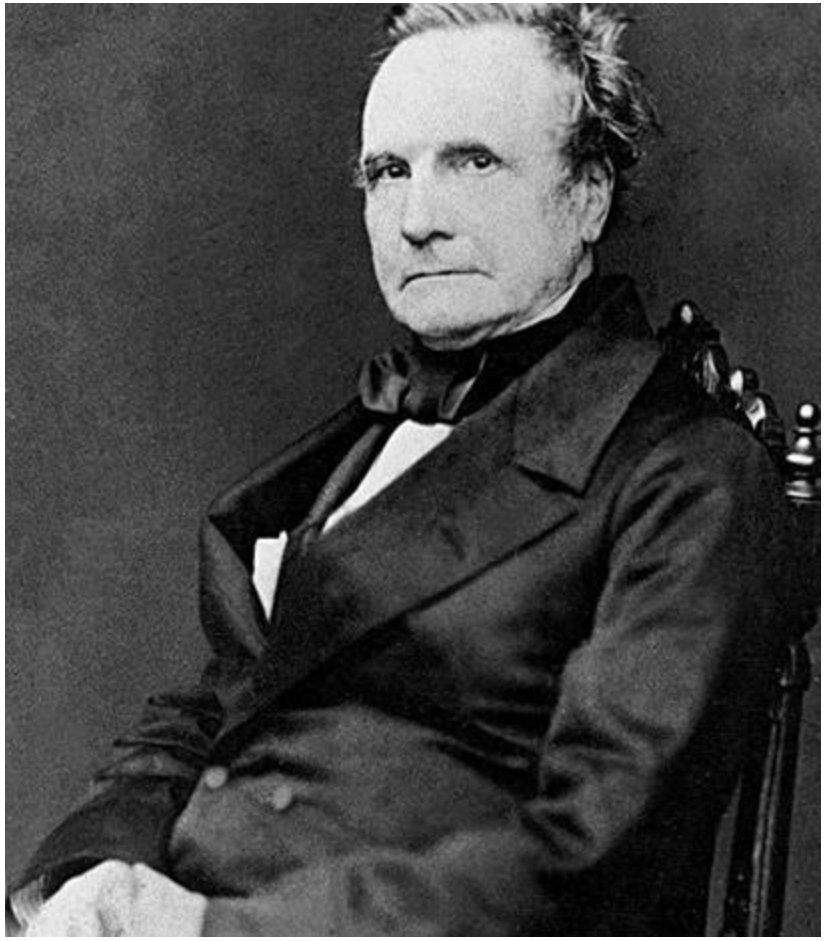
Charles Babbage

- Matemático Inglês
- Projetou computadores mecânicos
  - A máquina diferencial, não chegou a ser terminada por Babbage mas foi reconstruida em 1991
  - Máquina analítica (nunca construída)



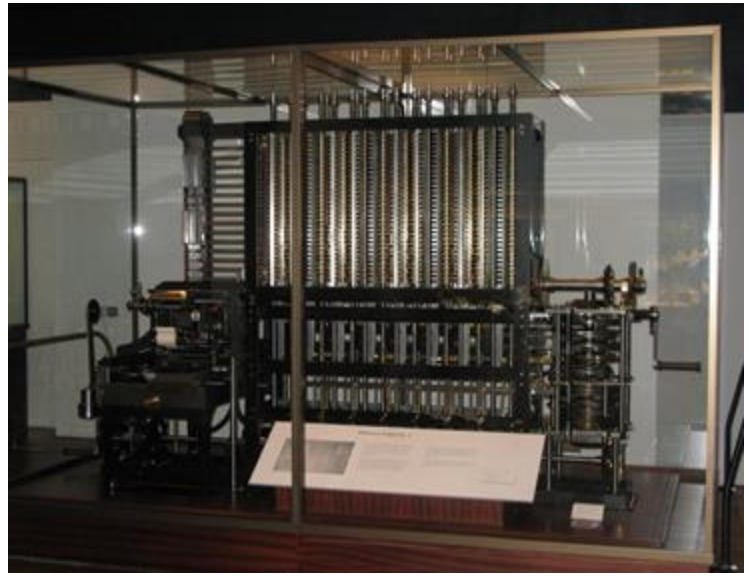
Charles Babbage

- Máquina Analítica
  - Tinha conceitos fundamentais de um computador moderno:
    - Unidade de processamento (o Mill)
    - Memória (o Store)
    - Programável com cartões perfurados
    - Estruturas de controle: iteração, desvio condicional, pipelining e I/O



Ada e Charles

- Ada Lovelace e Charles Babbage ficaram amigos e trabalharam juntos em muitas das ideias de Babbage.
- Ada descreveu um algoritmo em 1843 para calcular números de Bernoulli usando a máquina analítica.
  - Não chegou a ser testado, afinal a máquina não existia de fato.
  - Mas é considerado o primeiro programa publicado.



Ada e Charles

The bounds of *arithmetic* were however outstepped the moment the idea of applying the cards had occurred; and the Analytical Engine does not occupy common ground with mere "calculating machines." ... In enabling mechanism to combine together *general* symbols in successions of unlimited variety and extent, a uniting link is established between the operations of matter and the abstract mental processes of the *most abstract* branch of mathematical science.

A.A.L.

Plankalkül de Zuse

- 1936: construiu um computador mecânico, na sala de estar de seus pais, em Berlim:
  - O Z1
- Era um ábaco mecânico, controlado por pinos de metal e correias.
- Programável via fitas perfuradas
- Números de ponto flutuante, em binário, com expoente explícito.

Plankalkül de Zuse

- Projetado em **1945**, mas apenas **publicado em 1972**
- Algumas de suas ideias foram **reinventadas cerca de 15 anos depois**...
- **Nunca implementado** em sua época.
- Introduziu conceitos avançados:
  - Estruturas de dados avançadas
  - Números em ponto flutuante
  - Arrays
  - Registros
- Recurso interessante: **expressões matemáticas semelhantes a assertivas** (condições que devem ser verdadeiras).
- O documento tinha **49 páginas de algoritmos para jogar xadrez**!
- Do alemão:
  - *Kalkül* = sistema formal
  - *Plankalkül* = sistema formal para planejamento

Plankalkül de Zuse

- A seguinte sentença de atribuição de exemplo:

```
A[5] = A[4] +1
```

- A linha rotulada *V* é para os índices

- *S* é para os tipos de dados.
- 1.n significa um inteiro de n bits:

```
    |  A + 1 =>     A
V   |  4            5
S   |  1.n          1.n
```

Pseudocódigos

- **Anos 1940 e início dos 1950**
- Não tinham o mesmo significado que hoje.
- Na época não existia linguagem de máquina de alto nível, nem mesmo assembly.
- Programação era feita diretamente em **código de máquina**.
- Problemas de usar código de máquina:
  - **Difícil de modificar** e sujeito a muitos erros (endereçamento absoluto).
  - **Difícil de programar** em geral.
  - **Baixa legibilidade**.
  - **Sem suporte a números em ponto flutuante**.

Pseudocódigos

- Computadores: **caros, lentos, pouco confiáveis, com memória mínima**.
- **Não existiam** linguagens de alto nível nem de montagem.
- Programação feita em **linguagem de máquina** (códigos numéricos).
  - Ex.: 14 = ADD.
- Programas difíceis de ler, modificar e depurar.

Pseudocódigos

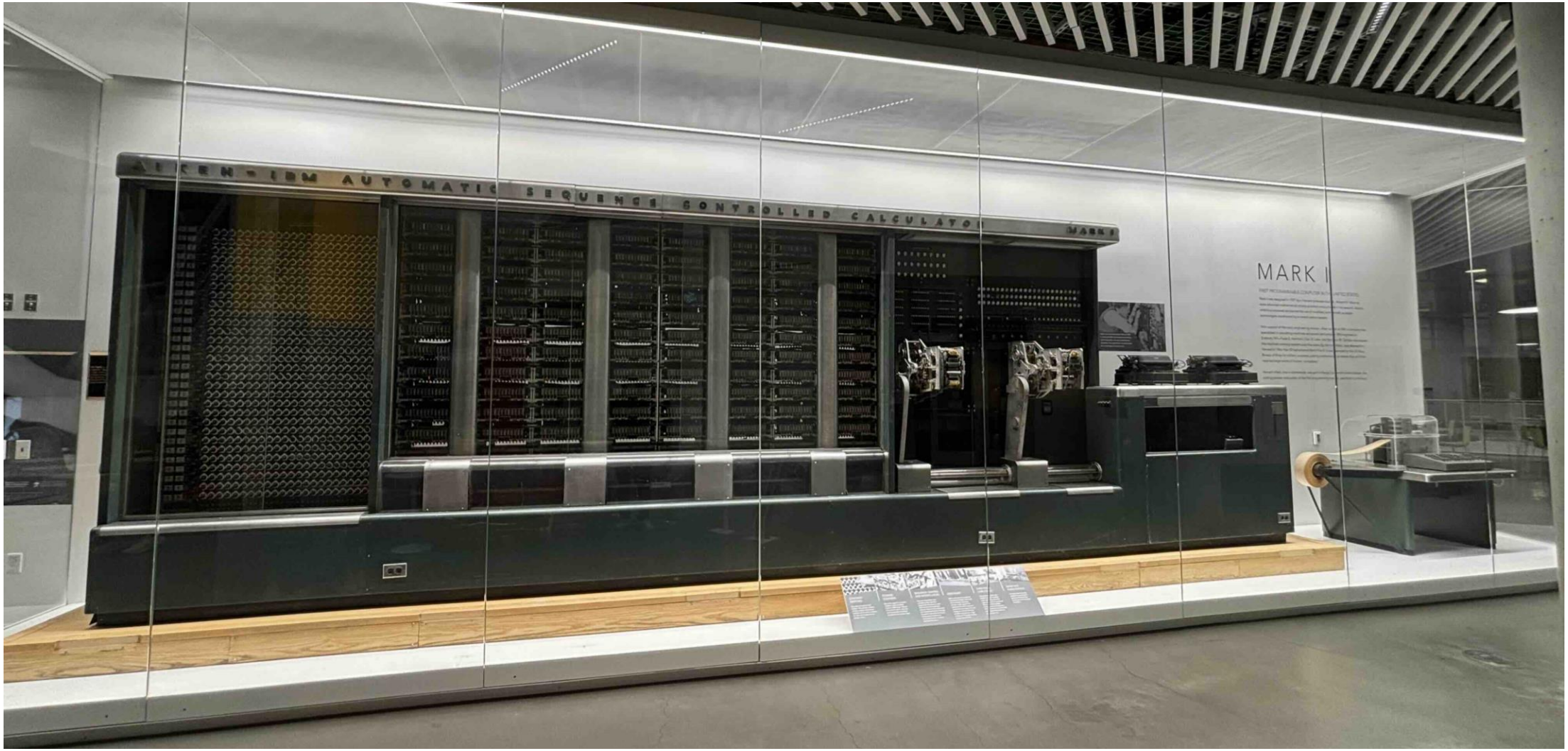
- Instruções usavam **endereços fixos da memória**.
- Inserir instrução no meio → desloca endereços → **todas as referências seguintes ficam erradas**.
- Consequência: **manutenção tediosa e propensa a erros**.
- Essas dificuldades motivaram depois o desenvolvimento de **montadores** e **linguagens de nível mais alto**.

Pseudocódigos

In the early years of programming languages, the most frequent phrase we heard was that the only way to program a computer was in octal. Of course a few years later a few people admitted that maybe you could use assembly language.... I have here a copy of the manual for **Mark I**. I think most of you would be totally flabbergasted if you were faced with programming a computer, using a **Mark I** manual. All it gives you are the codes. From there on you're on your own to write a program. We were not programmers in those days. The word had not yet come over from England. We were "coders."

Rear Admiral Dr. Grace Murray Hopper

MARK I



MARK I


MARK I
FIRST PROGRAMMABLE COMPUTER IN THE UNITED STATES
Mark I was designed in 1937 by a Harvard graduate student, Howard H. Aiken to solve advanced mathematical physics problems encountered in his research. Aiken's ambitious proposal envisioned the use of modified, commercially available technologies coordinated by a central control system.
With support of Harvard's engineering division, Aiken turned to IBM, a company that specialized in calculating machines and punch card systems. IBM engineers in Endicott, NY—Frank E. Hamilton, Clair D. Lake, and Benjamin M. Durfee—developed the machine's working systems over five years. By the time Mark I was delivered to Harvard in 1944, the US had entered World War II. It was operated by the US Navy Bureau of Ships for military purposes, solving mathematical problems that until then required large teams of human "computers."
Howard Aiken, now a commander, was put in charge. Lieutenant Grace Hopper, the coding pioneer and author of the first programming manual, was third in command.

Necessidades

- **Ponto flutuante:** os programadores tinham que controlar o expoente manualmente (estilo babilônico).
- **Endereçamento relativo:** os programadores mantinham cadernos de sub-rotinas, mas os códigos precisavam ser ajustados à mão para os endereços absolutos.
- Ajuda para indexação de vetores (arrays).
- Algo mais fácil de lembrar do que os códigos de operação em octal.

Necessidades

- **Assemblers**
- Ferramentas de programação:
  - **Short Code**, John Mauchly, 1949 (interpretado)
  - **A-0, A-1, A-2**, Grace Hopper, 1951–1953 (como bibliotecas de macros)
  - **Speedcoding**, John Backus, 1954 (interpretado)
- As pessoas começaram a perceber que economizar o tempo do programador era importante.

IBM 704 e Fortran

IBM 704 e Fortran

- **Fortran 0 (1954):** não implementado.
- Relatório: *“The IBM Mathematical FORmula TRANslating System: FORTRAN”*
- Time liderado por John Backus na IBM
- **Fortran I (1957):**
  - Projetado para o novo **IBM 704**.
  - Considerado um dos **maiores avanços individuais da computação**.
  - Impulsionou o desenvolvimento de linguagens de programação de **alto nível**.

IBM 704 e Fortran

- **Fortran I (1957):**
  - Projetado para o novo **IBM 704**, que tinha **registradores de índice** e **hardware para operações em ponto flutuante**.
- Isso levou à ideia de **linguagens compiladas**:
  - Antes, os **interpretadores eram tolerados** principalmente porque não havia suporte a **operações em ponto flutuante no hardware**.
  - Tudo precisava ser feito em **software**, o que era lento.

IBM 704 e Fortran

- **Fortran I (1957):**
  - **Ambiente de desenvolvimento:**
    - Computadores eram **pouco confiáveis**.
    - As aplicações eram principalmente **científicas**.
    - Não havia **metodologias ou ferramentas de programação**.
    - A **eficiência da máquina** era a maior preocupação.

Fortran I

- **Primeira versão implementada do Fortran**
  - Nomes podiam ter até **6 caracteres** (apenas 2 na versão 0).
  - Estrutura de repetição com contagem pós-teste (**DO**).
  - Entrada e saída **formatadas (I/O)**.
  - **Subprogramas definidos pelo usuário**.
  - Estrutura de seleção de **três vias** (*arithmetic IF*).
  - Não havia declarações explícitas de tipos de dados:
    - Nomes começando com **I, J, K, L, M, N → inteiros** (implicitamente).
    - Outros nomes → ponto flutuante.

Fortran I

- **Primeira versão implementada do FORTRAN**
  - O compilador foi lançado em **abril de 1957**, após **18 anos-homens de esforço**.
  - Programas com mais de **400 linhas raramente compilavam corretamente**, principalmente devido à **baixa confiabilidade do IBM 704**.
  - O código gerado era **muito rápido**.
  - Rapidamente se tornou **amplamente utilizado**.

Fortran IV

- Evoluiu entre **1960–1962**
  - **Declarações explícitas de tipos**
  - Estrutura de seleção **lógica**
  - **Nomes de subprogramas** podiam ser passados como parâmetros
  - Tornou-se **padrão ANSI em 1966**

Fortran 77

- Tornou-se o **novo padrão em 1978**
  - Suporte a **manipulação de strings de caracteres**
  - Controle lógico de laços (**loop control**)
  - Inclusão da estrutura **IF–THEN–ELSE**

Fortran 90

- Mudou **significativamente** em relação ao Fortran 77:
  - **Módulos**
  - **Vetores dinâmicos (arrays dinâmicos)**
  - **Ponteiros**
  - **Recursão**
  - Estrutura **CASE**
  - **Verificação de tipos de parâmetros**

Últimas versões Fortran

- **Fortran 95** – pequenas adições e algumas remoções.
- **Fortran 2003** – passou a suportar **Programação Orientada a Objetos (OOP)**.
- **Resumo:**
  - Fortran é reconhecido como a **primeira linguagem de alto nível**.
  - Evoluiu com recursos modernos.
  - **Ainda é usado atualmente**, especialmente em aplicações científicas e de engenharia.

Últimas versões Fortran

- **Fortran 95** – pequenas adições e algumas remoções.
- **Fortran 2003** – passou a suportar **Programação Orientada a Objetos (OOP)**.
- **Resumo:**
  - Fortran é reconhecido como a **primeira linguagem de alto nível**.
  - Evoluiu com recursos modernos.
  - **Ainda é usado atualmente**, especialmente em aplicações científicas e de engenharia.

Últimas versões Fortran

```
! Fortran 95 Example program
!  Input:  An integer, List_Len, where List_Len is less
!          than 100, followed by List_Len-Integer values
!  Output: The number of input values that are greater
!          than the average of all input values
Implicit none
Integer Dimension(99) :: Int_List
Integer :: List_Len, Counter, Sum, Average, Result
Result= 0
Sum = 0
Read *, List_Len
If ((List_Len > 0) .AND. (List_Len < 100)) Then
! Read input data into an array and compute its sum
   Do Counter = 1, List_Len
      Read *, Int_List(Counter)
      Sum = Sum + Int_List(Counter)
                End Do
             ! Compute the average
                Average = Sum / List_Len
             ! Count the values that are greater than the average
                Do Counter = 1, List_Len
                   If (Int_List(Counter) > Average) Then
                      Result = Result + 1
                   End If
                End Do
             ! Print the result
                Print *, 'Number of values > Average is:', Result
             Else
                Print *, 'Error - list length value is not legal'
             End If
             End Program Example
```

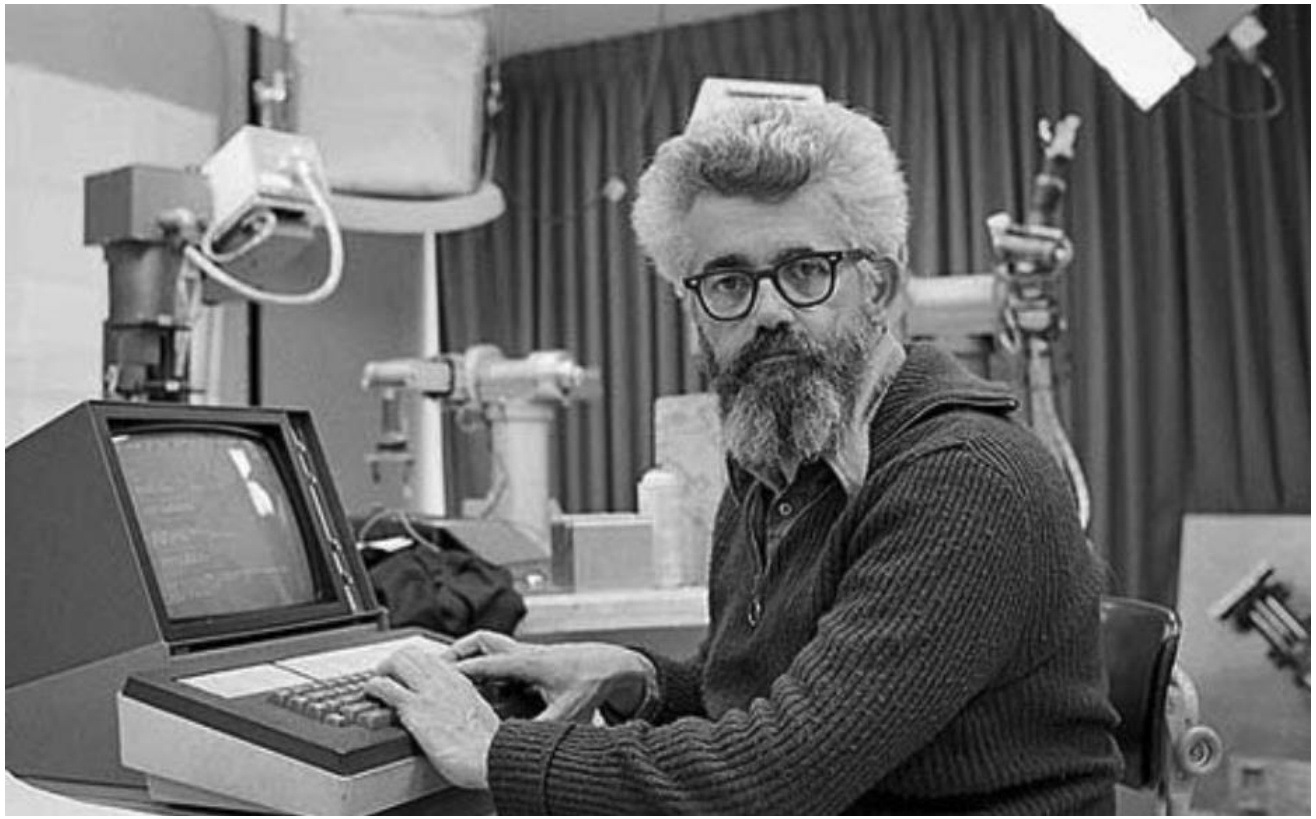
COBOL – Common Business Oriented Language

- Desenvolvida em 1959 pelo Departamento de Defesa dos EUA e fabricantes de computadores
- Tornou-se padrão para **aplicações comerciais**
- Independente de máquina (portável entre diferentes sistemas)
- **Contribuições**
  - Código mais **legível** (“English-like”)
  - **Estruturas de dados heterogêneas**
  - **Registros (records)**
- Ainda hoje é usado em sistemas bancários, de governo e grandes empresas.

```
 IDENTIFICATION DIVISION.
   PROGRAM-ID. HELLO-WORLD.
*
 ENVIRONMENT DIVISION.
*
 DATA DIVISION.
*
 PROCEDURE DIVISION.
 PARA-1.
   DISPLAY "Hello, world.".
*
      EXIT PROGRAM.
   END PROGRAM HELLO-WORLD.
```

Programação funcional: Lisp

- Projetada em 1958 no Instituto de Tecnologia de Massachusetts (MIT) por McCarthy



Programação funcional: Lisp

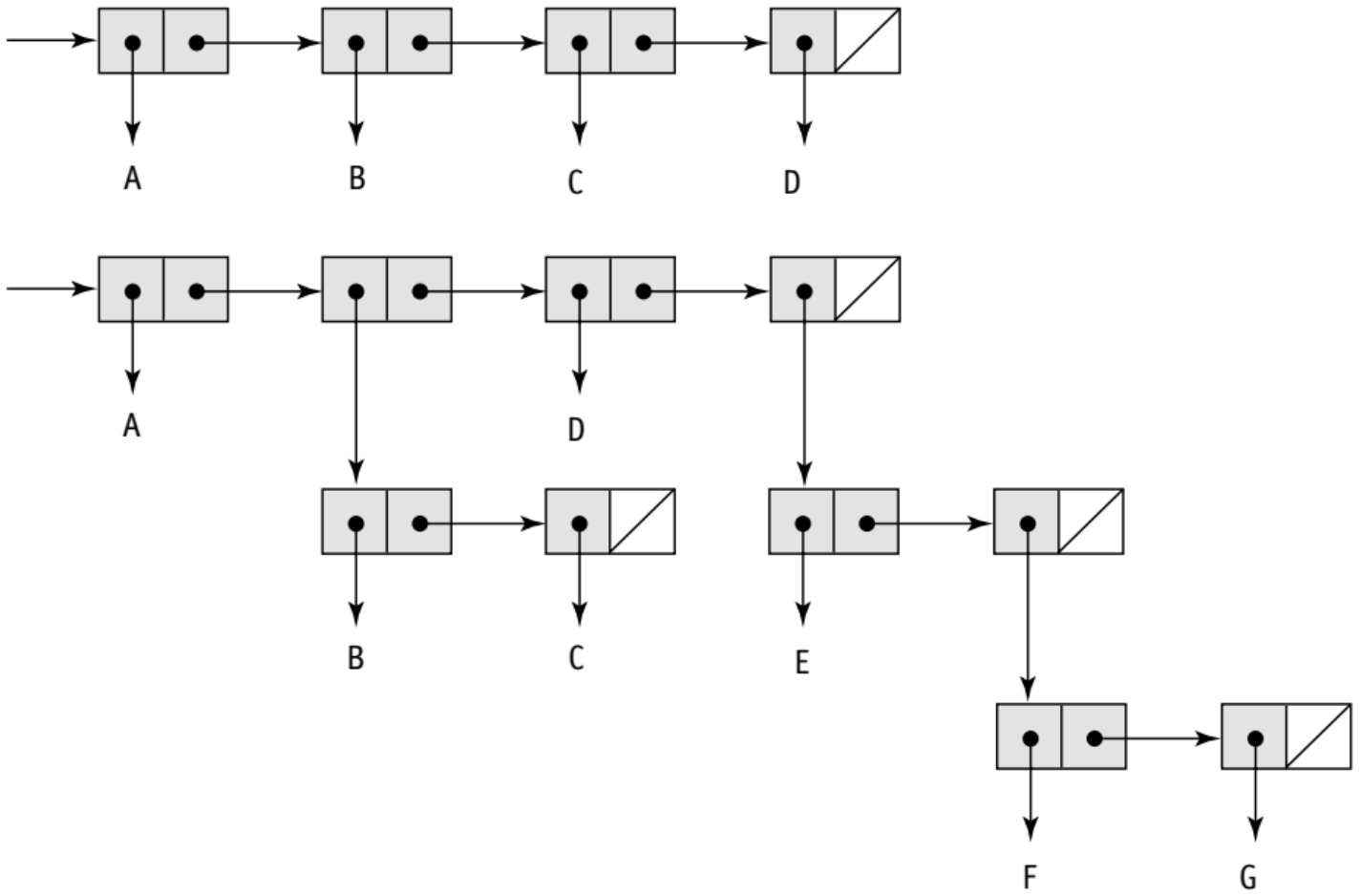
- A pesquisa em **Inteligência Artificial** precisava de uma linguagem.
  - Crescente **interesse em IA na metade dos anos 1950**.
  - Interesse de diversas áreas:
    - **Linguistas** e processamento de linguagem natural.
    - **Psicólogos** e estudos sobre o **cérebro**.
    - **Matemáticos**.
    - Objetivo de **mecanizar processos inteligentes**.

Programação funcional: Lisp

- A pesquisa em IA precisava de uma linguagem para **processar dados em listas** (em vez de arrays).
- Por que listas?
  - Mais **flexíveis que arrays**.
  - Permitem dados mais **heterogêneos**.
  - Suportam **hierarquias**.
  - Facilitam **decisão em árvores** e **recursão**.
- **Computação simbólica** (em vez de numérica)
  - “Simbólico” no sentido de trabalhar com **símbolos** e relações abstratas.
  - Ex.: relações entre itens em uma lista → mais abstratos como **cores, lugares, pessoas, partes hierárquicas de objetos, coisas que você lembra**, etc.
  - Podem ter **valor semântico**.

Programação funcional: Lisp

- Apenas dois tipos de dados: átomos e listas
- (A B C D)
- (A (B C) D (E (F G)))



Programação funcional: Lisp

- Muito diferente das **linguagens imperativas**.
- Todos os cálculos feitos **aplicando funções a argumentos**.
- **Atribuições e variáveis não são necessárias**.
- Não há **loops** → tudo é feito com **recursão**.
- Sintaxe muito diferente (compare **C** com **Lisp** e sua simplicidade).
- Baseado formalmente no **cálculo lambda**, introduzido nos anos **1930** para investigar:
  - Definição de funções
  - Aplicação de funções
  - Recursão

Programação funcional: Lisp

- Foi o **pioneiro da programação funcional**:
  - Não havia necessidade de **variáveis ou atribuições**.
  - Controle feito por **recursão** e **expressões condicionais**.
- O **Lisp original** é chamado de **“Pure Lisp”** porque era **puramente funcional**.
- Dominou a área de **Inteligência Artificial por 25 anos** e ainda é relevante.
  - Hoje há um **novo interesse em IA e aprendizado de máquina**.
  - Outras linguagens **imperativas** também são usadas, como o **Python**.

Programação funcional: Lisp

- **Common Lisp** (adicionou recursos de linguagem imperativa) e **Scheme** são dialetos contemporâneos do Lisp (detalhes mais adiante).
- **ML, Miranda e Haskell** são linguagens relacionadas.
- **ML** possui alguns elementos **imperativos** e não usa parênteses.
- **Haskell** introduziu a ideia de **avaliação preguiçosa (lazy evaluation):**
  - As expressões **só são avaliadas quando o valor é realmente necessário**.
  - Melhora eficiência em alguns casos, pois evita cálculos desnecessários.

Programação funcional: Lisp

```
;  LISP Example function
;  The following code defines a LISP predicate function
;  that  takes two lists as arguments and returns True
;  if the two lists are equal, and NIL (false) otherwise
   (DEFUN equal_lists (lis1 lis2)
    (COND
      ((ATOM lis1) (EQ lis1 lis2))
      ((ATOM lis2) NIL)
      ((equal_lists (CAR lis1) (CAR lis2))
            (equal_lists (CDR lis1) (CDR lis2)))
(T NIL) )

)
```

ALGOL (Algorithmic Language)

- Em 1957, as linguagens estavam se proliferando
- Nos EUA, fabricantes de computadores estavam desenvolvendo linguagens específicas de plataforma, como o **Fortran** da IBM
- Na Europa, várias linguagens estavam sendo projetadas por diferentes grupos de pesquisa: **Plankalkül** e outros
- O Algol foi criado para interromper essa proliferação
- Seria a **linguagem universal, internacional e independente de máquina** para expressar algoritmos científicos
- Em 1958, um comitê internacional foi formado para desenvolver o projeto

ALGOL

- Eventualmente, três grandes versões foram projetadas:
  - **Algol 58**
  - **Algol 60**
  - **Algol 68**
- Desenvolvidos por comitês internacionais cada vez maiores

ALGOL

- Praticamente **todas as linguagens após 1958** usaram ideias pioneiras dos projetos Algol:
  - Declarações compostas: **begin** *statements* **end**
  - Estrutura lexical de formato livre
  - Definição de sintaxe em **BNF (Backus-Naur Form)**
  - Variáveis locais com escopo de bloco
  - Tipagem estática com declarações explícitas de tipo
  - If-then-else aninhado
  - Chamada por valor (e chamada por nome)
  - Sub-rotinas recursivas e expressões condicionais (ex: Lisp)
  - Arrays dinâmicos
  - Procedimentos de primeira classe
  - Operadores definidos pelo usuário

ALGOL

- As primeiras linguagens usavam controle orientado a rótulos (labels):

```
GO TO 27
IF (A-B) 5,6,7
```

- As linguagens Algol tinham bom controle em nível de frases, como o **if** e **while** que vimos em Java, além de **switch, for, until**, etc.
- Um debate sobre os méritos relativos começou a esquentar.
- A famosa carta de Edsger Dijkstra em 1968, ***"Go to statement considered harmful"***, propôs eliminar completamente o controle orientado a rótulos.

ALGOL

- Usar controle em nível de frases no lugar de rótulos passou a ser chamado de **programação estruturada**.
- Houve um longo debate: muitos programadores achavam difícil, no início, trabalhar sem rótulos.
- Hoje, a revolução já aconteceu:
  - Algumas linguagens (como **Java**) eliminaram o goto.
  - Outras (como **C++**) ainda possuem, mas raramente é usado.
- Essa revolução foi desencadeada (ou ao menos impulsionada) pelos projetos Algol.

ALGOL

- Os projetos do Algol evitaram casos especiais.
- Estrutura lexical de formato livre.
- Sem limites arbitrários:
  - Qualquer número de caracteres em um nome.
  - Qualquer número de dimensões em um array.
- E a **ortogonalidade**: toda combinação significativa de conceitos primitivos era permitida — sem combinações proibidas que o programador tivesse que decorar.

ALGOL

	Integers	Arrays	Procedures
Passing as a parameter			
Storing in a variable			
Storing in an array			
Returning from a procedure			

- Cada combinação **não** permitida é um caso especial que deve ser lembrado pelo programador.
- Com o Algol 68, todas as combinações acima são legais.
- Apenas uma amostra de sua **ortogonalidade** — poucas linguagens modernas levam esse princípio tão longe quanto o Algol.

ALGOL

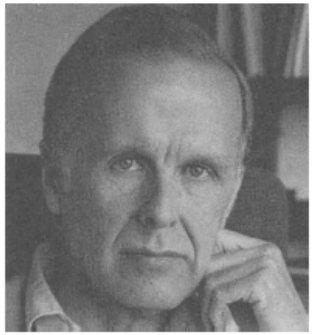
- As linguagens Algol não foram tão amplamente usadas quanto se esperava.
  - **Algol 58** evoluiu para Jovial.
  - **Algol 60** foi usado para publicação de algoritmos, e implementado e usado de forma razoavelmente ampla fora dos EUA.
- Algumas razões possíveis:
  - Negligenciaram **I/O** (entrada e saída).
  - Foram consideradas complicadas e difíceis de aprender.
  - Não tiveram patrocínio corporativo (a **IBM** escolheu manter o Fortran).

John Backus

- Turing Award de 1977
- **Fortran**
- Algol 58 e 60
- **Backus-Naur Form (BNF)**
- FP (uma linguagem puramente funcional)

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

613

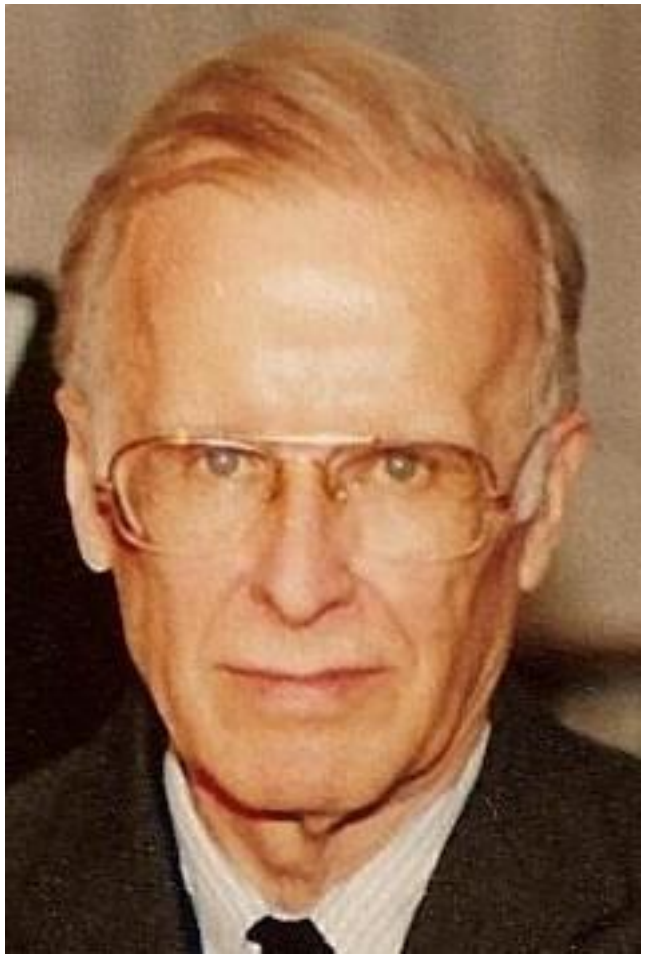
Communications of the ACM

August 1978 Volume 21 Number 8

John Backus

"My point is this: while it was perhaps natural and inevitable that languages like FORTRAN and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. It is unfortunate because their long - standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computation power."

THE HISTORY OF FORTRAN I, II, AND III
John Backus, 1978



Smaltalk

Antes do Smaltalk: Simula

- Kristen Nygaard e Ole-Johan Dahl, Norwegian Computing Center, 1961
- **Simula I**: uma extensão especial de **Algol** para programação de simulações: aviões em um aeroporto, clientes em um banco, etc.
- **Simula 67**: uma linguagem de uso geral com classes, objetos, herança
- Usava **co-rotinas** em vez de métodos

Pascal

- Desenvolvida por **Niklaus Wirth** em 1971 (ex-membro do comitê do ALGOL 68)
- Projetada para o **ensino de programação estruturada**
- Pequena, simples, nada realmente novo
- Maior impacto foi no ensino de programação
- Do meio dos anos 1970 até o fim dos anos 1990, foi a linguagem mais utilizada para o ensino de programação

Pascal

```
program temperature(output) ;

{ Program to convert temperatures from
 Fahrenheit to Celsius. }

const
    MIN = 32 ;
    MAX = 50 ;
    CONVERT = 5 / 9 ;

var
    fahren: integer ;
    celsius: real ;

begin
    writeln('Fahrenheit     Celsius') ;
    writeln('----------     -------') ;
    for fahren := MIN to MAX do begin
        celsius := CONVERT * (fahren - 32) ;
        writeln(fahren: 5, celsius: 18: 2) ;
 end ;
end.
```

C

- Originalmente projetada para programação de sistemas (nos Bell Labs por **Dennis Ritchie** em 1972)
- Evoluiu principalmente de **BCPL** e **B** (não tipada), mas também de **ALGOL 68** (com for, switch, ponteiros)
- Conjunto poderoso de operadores, mas com verificação de tipos fraca
- Inicialmente se espalhou através do **UNIX** (gratuito; uso disseminado)
- Muitas áreas de aplicação

C

- Foi o padrão por muito tempo: livro de **Kernighan e Ritchie (1978)**
- Padrão ANSI em **1989**

```
#include <stdio.h>

inline float convert(float f)  {
    return ((5.0/9.0) * (f - 32));
}

int main()  {
    float f;
    for(f = -40; f <= 220; f += 10) {
      printf("%f degrees fahrenheit = %f degrees celsius.\n", f, convert(f));
    }
}
```

Prolog

- **Programação Baseada em Lógica: Prolog**
- Desenvolvido por **Colmerauer e Roussel** (Universidade de Aix-Marseille), com ajuda de **Kowalski** (Universidade de Edimburgo) no início dos anos 1970
- Baseado em **lógica formal**
- **Não-procedural**
- Pode ser resumido como um **sistema de banco de dados inteligente** que usa um processo de inferência para deduzir a veracidade de consultas feitas
- Altamente ineficiente, com **pequenas áreas de aplicação**

Prolog

```
mother(joanne,jake).
father(vern,joanne).


grandparent(X,Z)  :=
  parent(X,Y),
  parent(Y,Z).
```

```
Query: father(bob,darcie).
```

Smaltalk

- Alan Kay, Xerox PARC, 1972
- Inspirado por Simula, etc.
- Smalltalk é **mais orientado a objetos** do que a maioria de seus descendentes populares
- Tudo é um objeto: variáveis, constantes, registros de ativação, classes, etc.
- Toda computação é realizada por objetos que enviam e recebem mensagens

Smaltalk

- As linguagens **Simula e Smalltalk** inspiraram uma geração de linguagens orientadas a objetos
- Smalltalk ainda tem uma comunidade de usuários pequena, mas ativa
- A maioria das linguagens OO posteriores se concentram mais em **eficiência de execução**:
  - A maioria usa **tipagem estática** (Smalltalk usa dinâmica)
  - A maioria inclui **tipos primitivos não-objetos** além de objetos

C++

- **Combinando Programação Imperativa e Orientada a Objetos**
- Desenvolvida nos Bell Labs por Bjarne Stroustrup em 1980
- Evoluiu a partir de C e SIMULA 67
- Recursos para programação orientada a objetos, parcialmente herdados do SIMULA 67
- Fornece tratamento de exceções

C++

- Uma linguagem grande e complexa, em parte porque suporta tanto programação procedural quanto orientada a objetos
- Cresceu rapidamente em popularidade, junto com a OOP (programação orientada a objetos)
- Compiladores bons e baratos; compatível retroativamente com C
- Padrão ANSI aprovado em novembro de 1997
- Versão da Microsoft (lançada com o .NET em 2002): Managed C++

Objective C

- Outro híbrido que combina programação imperativa e orientada a objetos (OOP)
- Inicialmente consistia em C mais classes e troca de mensagens como no SmallTalk
- Steve Jobs fundou a NeXT, que usava Objective-C; depois a Apple comprou a NeXT e adotou o Objective-C
- Linguagem do **Mac OS X** e do software do iPhone, o que aumentou bastante sua popularidade
- Hoje substituída pelo **Swift** (mais seguro, por exemplo, remove o gerenciamento inseguro de ponteiros)

Java

- James Gosling, Sun Microsystems
- 1991: Oak – uma linguagem para computadores ubíquos em tecnologia de consumo em rede
  - Como C++, mas menor e mais simples
  - Mais segura e fortemente tipada
  - Mais independente de plataforma
- 1995: renomeada Java, redirecionada para a Web
  - Incorporada em navegadores
  - Conteúdo ativo independente de plataforma para páginas web

Java

- Baseada em C++
- Suporta POO
- Possui referências, mas não ponteiros
- Coleta de lixo (ex. Lisp)
- Concorrência (ex. Mesa)
- Pacotes (ex. Modula)
- Mas nada realmente novo: foi concebida para ser uma **linguagem de produção**, não de pesquisa

Linguagens de Script para a Web

- **JavaScript (meados dos anos 1990)**
  - Começou na Netscape, mas depois se tornou uma joint venture da Netscape e Sun Microsystems
  - Uma linguagem de script embutida em HTML no lado do cliente, frequentemente usada para criar documentos HTML dinâmicos
  - Puramente interpretada
  - Relacionada ao Java apenas por ter sintaxe semelhante

Linguagens de Script para a Web

- **PHP (1994)**
  - **PHP: Hypertext Preprocessor**, projetado por Rasmus Lerdorf para fornecer uma ferramenta de rastreamento de visitantes em seu site
  - Uma linguagem de script embutida em HTML no lado do servidor, frequentemente usada para processamento de formulários e acesso a banco de dados pela Web
- **Outras linguagens de script:** Perl, Lua
- **Ruby (1996);** popularizado pelo **Ruby on Rails (2004)**

Python

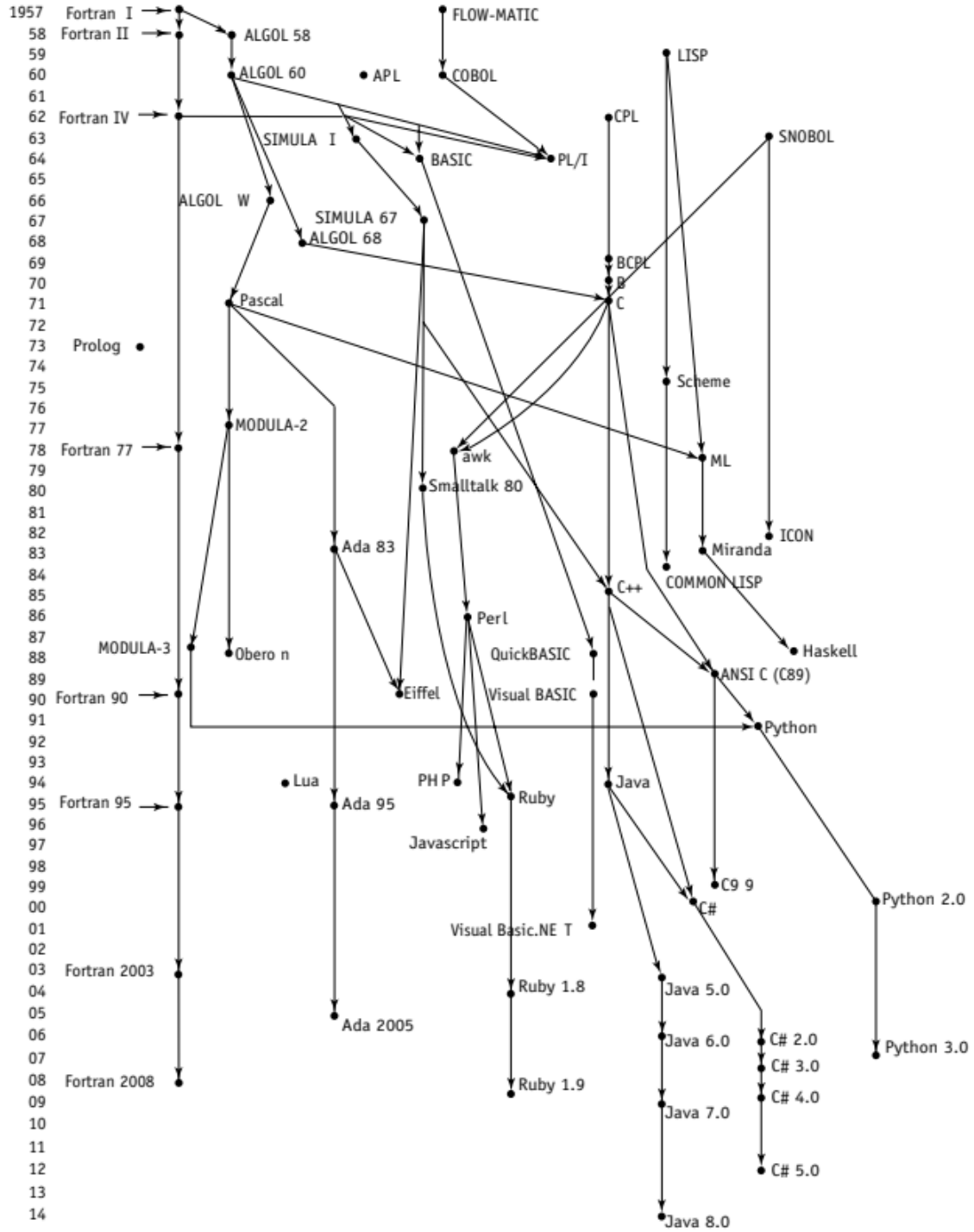
- Nomeado em referência ao grupo de comédia *Monty Python*
- Uma linguagem de script interpretada e orientada a objetos
- Tipagem verificada, mas dinamicamente tipada
- Usada inicialmente para programação **CGI (Common Gateway Interface)**; conjunto de padrões que definem como a informação é trocada entre o servidor web e um script personalizado
- Mais recentemente ganhou destaque em outras áreas, como **aprendizado de máquina** e **computação científica**

Turing Award

- Alguns pioneiros de linguagens de programação que ganharam o **Prêmio Turing**:

Alan Perlis, **John McCarthy**, **Edsger Dijkstra**, Donald Knuth, Dana Scott, **John Backus**, Robert Floyd, Kenneth Iverson, C.A.R. Hoare, Dennis Ritchie, Niklaus Wirth, John Cocke, Robin Milner, **Kristen Nygaard, Ole-Johan Dahl**, **Alan Kay, Peter Naur,** Frances Allen, and Barbara Liskov

Genealogia das principais linguagens de programação de alto nível.



Paradigmas de Linguagens de Programação

Histórico das Linguagens de Programação