ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

LINGUAGENS DE PROGRAMAÇÃO

1. Introdução às Linguagens de Programação
2. Paradigmas
3. Modelos de Implementação
4. Evolução das LP

Motivos para estudar os conceitos de linguagens de programação

- Aumento da capacidade de expressar idéias
  - Limitações de uma linguagem
    - Tipos de estrutura de dados
    - Tipos de estrutura de controle
    - Tipos de abstrações que podem ser utilizadas
  - Assim, as formas de algoritmos também são limitadas
  - Conhecimento de recursos de linguagens de programação reduz essas limitações

Motivos para estudar os conceitos de linguagens de programação

- Aumento da capacidade de expressar idéias
  - Os programadores aumentam seu poder de expressão aprendendo novas construções de linguagens
    - Mesmo que a linguagem utilizada não possua a capacidade aprendida.
    - Geralmente, facilidades de uma linguagem podem ser simuladas em outra.

Motivos para estudar os conceitos de linguagens de programação

- Maior conhecimento para escolha de linguagens apropriadas
  - Não conhecimento dos conceitos de linguagens de programação
    - Utilização da linguagem que está mais familiarizada mesmo que não seja adequada ao novo projeto.
  - O conhecimento dos recursos fornecidos pelas linguagens permite que se faça uma escolha mais consciente

Motivos para estudar os conceitos de linguagens de programação

- Maior habilidade para aprender novas linguagens
  - Compreensão dos conceitos fundamentais das linguagens
    - Facilita a percepção de como estes são incorporados ao projeto da linguagem aprendida.
    - Facilita a leitura e o entendimento de manuais e do "marketing" relacionado a linguagens e compiladores

Motivos para estudar os conceitos de linguagens de programação

- Entender melhor a importância da implementação
  - Capacidade de utilizar uma linguagem de forma mais inteligente, como ela foi projetada.
  - Facilita a detecção e correção de certos *bugs*
  - Visualização de como um computador executa várias construções da linguagem
    - Entendimento da eficiência relativa de construções alternativas a serem escolhidas

Motivos para estudar os conceitos de linguagens de programação

- Aumento da capacidade de projetar novas linguagens
  - Um exame crítico das linguagens de programação ajuda
    - No projeto de sistemas complexos
    - A examinar e avaliar esses produtos

Motivos para estudar os conceitos de linguagens de programação

- Avanço global da computação
  - Em alguns casos uma linguagem não se torna popular porque aqueles com capacidade de optar ainda não estavam familiarizados com os conceitos de linguagens de programação
  - Permanência no FORTRAN em detrimento do ALGOL 60 (mais elegante, melhores estruturas de controle, recursão e bloco) na década de 60.

Definição de Linguagem de Programação

- **Sintática**: Uma linguagem de programação é uma **notação** utilizada pelo programador para especificar ações a serem executadas por um computador

- **Semântica**: Uma linguagem de programação compreende um conjunto de **conceitos** que um programador usa para resolver problemas de programação

Domínios de Programação

- Aplicações científicas
- Aplicações comerciais
- Inteligência artificial
- Sistemas básicos
- Aplicações Internet

Domínios de Programação

- Computadores são utilizados em áreas bem diferentes
  - Controle de usinas elétricas nucleares
  - Armazenagem de registros de talões de cheques pessoais
- Têm sido desenvolvidas Linguagens de programação com metas muito diferentes

Domínios de Programação

- Aplicações científicas
  - Estrutura de dados simples
  - Computações aritméticas com ponto flutuante
  - Estrutura de dados comuns: array e matriz
  - Estruturas de controle comuns: laços de contagem e de seleções
  - FORTRAN
  - ALGOL 60 e suas descendentes

Domínios de Programação

- Aplicações comerciais
  - COBOL
  - Produz relatórios elaborados
  - Maneiras precisas de descrever e armazenar números decimais e dados caracteres
  - Capacidade de especificar operações aritméticas decimais

Domínios de Programação

- Inteligência Artificial
  - Uso de computações simbólicas em vez de numéricas
  - Linguagem funcional LISP
  - Programação lógica – linguagem Prolog

Domínios de Programação

- Programação de sistemas
  - Décadas de 60 e 70 alguns fabricantes de computadores desenvolveram linguagens de alto nível
    - IBM – linguagem PL/S
    - Digital – BLISS
    - Burroughs – Extended ALGOL
  - C, C++, Java, etc

Domínios de Programação

- Linguagens de Scripting
  - Lista de comandos (script) em arquivo para serem executados
  - sh (de shell – execução de funções utilitárias, gerenciamento e filtragem de arquivos)
  - awk (inicialmente geração de relatórios)
  - tcl e tk (Aplicações X Window)
  - JavaScript
  - Perl
    - Inicialmente combinou sh e awk
    - Popularizou-se com a Web (CGI)
  - JavaScript, PHP
  - Python
  - Lua

Critérios de Avaliação de Linguagens

- Legibilidade
- Redigibilidade
- Confiabilidade
- Custo
- Outros

Critérios de Avaliação de Linguagens

- **Legibilidade**
  - Facilidade com que os programas podem ser lidos e entendidos
  - Facilidade de manutenção é, em grande parte, determinada pela legibilidade dos programas

Critérios de Avaliação de Linguagens

- **Legibilidade**
  - **Simplicidade global**
    - Pequeno número de componentes básicos
    - Não possuir recursos múltiplos – mais de uma maneira de realizar uma operação particular
    - Sobrecarga de operador
  - **Simplicidade não significa necessariamente concisão**
    - A linguagem pode ser concisa mas usar muitos símbolos especiais
    - Exemplo: linguagens funcionais

Critérios de Avaliação de Linguagens

- **Legibilidade**
  - **Ortogonalidade**
    - **Um conjunto relativamente pequeno de construções primitivas pode ser combinado, em um número relativamente pequeno de maneiras, para construir as estrutura de controle e de dados da linguagem**
    - **O significado de um recurso de linguagem ortogonal é livre de contexto de sua aparência em um programa**

Critérios de Avaliação de Linguagens

- **Legibilidade**
  - **Ortogonalidade**
    - Torna a linguagem mais fácil de aprender
    - Diminui as restrições de programação
    - Muita ortogonalidade também pode causar problemas
    - Simplicidade
      - Combinação de um número relativamente pequeno de construções primitivas
      - Uso limitado do conceito de ortogonalidade

Critérios de Avaliação de Linguagens

- **Legibilidade**
  - **Instruções de controle**
    - **Restrição à instrução goto**
    - **Incluir instruções suficientes com o objetivo de eliminar a necessidade de utilização de goto**
    - **As instruções de controle devem ser bem projetadas**

Critérios de Avaliação de Linguagens

- **Legibilidade**
  - **Tipos de dados e estruturas**
    - **Facilidades para definir tipos e estrutura de dados auxilia a legibilidade**
  - **Sintaxe**
    - **Identificadores (Poucas restrições na escolha)**
      - **ANSI BASIC (1 Legra + 1 Dígito)**
      - **FORTRAN 77 (6 caracteres)**
    - **Palavras especiais (begin end x enf if x end loop)**
    - **Forma e significado (static em C)**
      - **Variável criada em tempo de compilação, quando definida em uma função**
      - **Inacessível de fora do arquivo, se não for definida em uma função**

Critérios de Avaliação de Linguagens

- **Redigibilidade**
  - Facilidade com que uma linguagem pode ser utilizada para criar programas para o domínio de problema escolhido
  - Simplicidade
    - Pequeno número de construções primitivas
  - Ortogonalidade
    - Um conjunto consistente de regras para combinar as construções da linguagem

Critérios de Avaliação de Linguagens

- **Redigibilidade**
  - Suporte a abstração (Habilidade para definir e usar estruturas e operações complexas de forma que muitos detalhes sejam ignorados)
    - Processos (subprogramas)
    - Dados (estruturas de dados)
  - Expressividade
    - Facilidade de expressar computação em programas pequenos (++ em C)
    - Expressividade x concisão
      - muito concisa: falta expressividade?
      - muito extensa: falta simplicidade?
  - Linguagens mais modernas:
    - incorporam apenas um conjunto básico de representações de tipos de dados e comandos
    - aumentam o poder de expressividade com bibliotecas de componentes
    - Exemplos: Pascal, C++ e Java

Critérios de Avaliação de Linguagens

- Confiabilidade
  - Verificação de tipos
  - Manipulação de exceções
  - Aliasing
    - Existência de métodos diferentes de acesso a uma mesma célula de memória
    - Mais de um apontador para a mesma variável
  - Legibilidade e Redigibilidade
    - Quanto mais fácil descrever e ler, maior a confiabilidade do programa

Critérios de Avaliação de Linguagens

- Custo
  - Treinamento de programadores
  - Escrever programas na linguagem
  - Compilar programas na linguagem
  - Executar programas
  - Sistema de implementação da linguagem
  - Custo da má confiabilidade
  - Manutenção dos programas

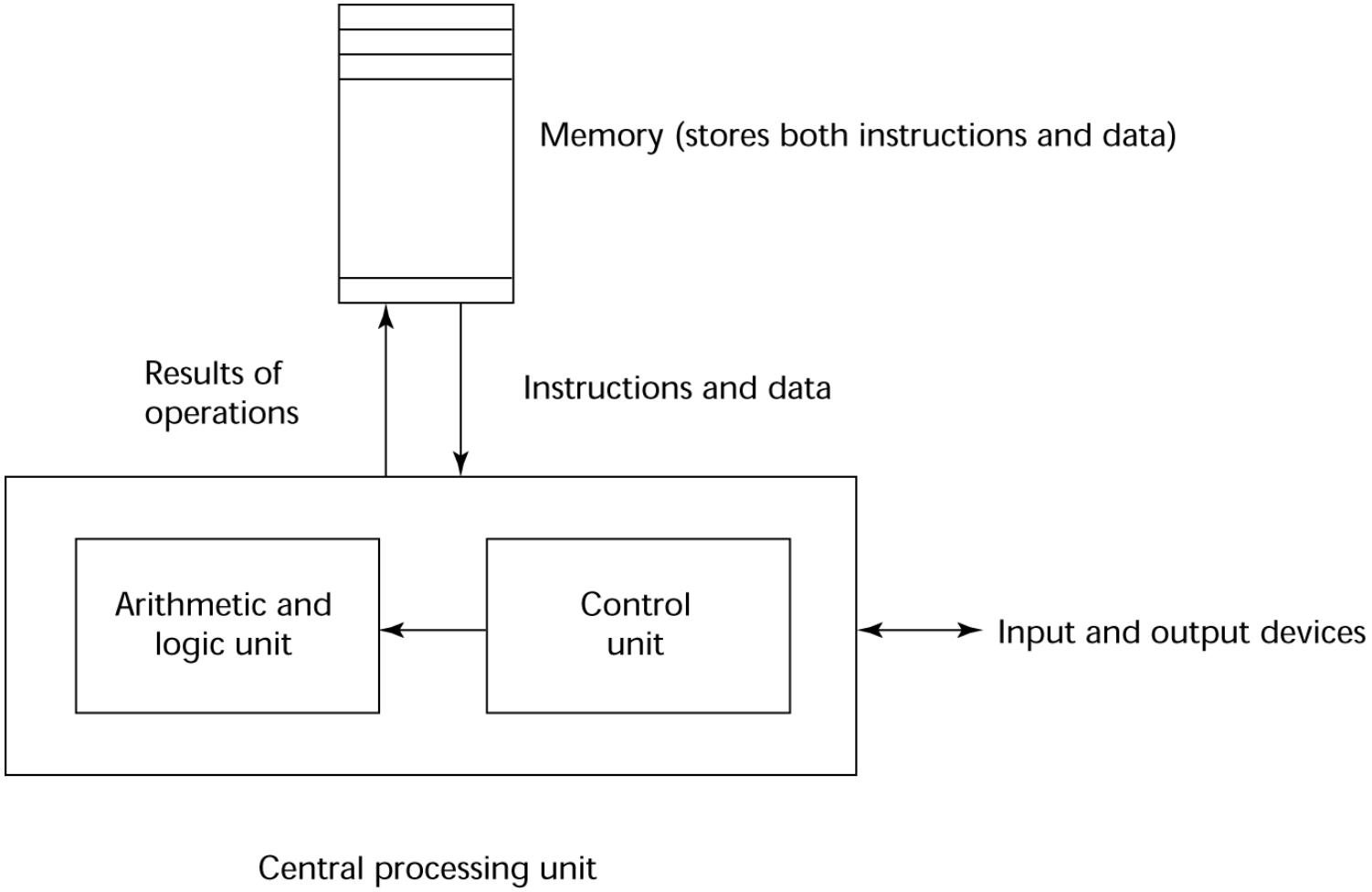
Critérios de Avaliação de Linguagens

- **Portabilidade (Multiplataforma):**
  - capacidade de um software rodar em diferentes plataformas sem a necessidade de maiores adaptações
  - Sem exigências especiais de hardware/software
  - Exemplo: aplicação compatível com sistemas Unix e Windows
- **Longevidade:**
  - ciclo de vida útil do software e o do hardware não precisam ser síncronos; ou seja, é possível usar o mesmo software após uma mudança de hardware

Principais Influências sobre o Projeto de Linguagens

1. Arquitetura de Computadores
   - Usa-se majoritariamente linguagens imperativas, pelo menos em parte, por conta da arquitetura de von Neuman
2. Metodologias de Programação
   - 1950-1960 (início): Aplicações simples, preocupação com a eficiência de máquina
   - Final dos anos 1960: Eficiência das pessoas torna-se importante, legibilidade, melhores estruturas de controle
   - Final dos anos 1970: Abstração de dados
   - Meados dos anos 1980: Programação Orientada a Objeto

Arquitetura de von Neuman



Categorias de Linguagem

1. Imperativas  foco
   - C, Pascal, Fortran, Cobol, etc
2. Funcionais
   - LISP, Scheme, ML, Haskell, etc
3. Lógicas
   - Baseadas em regras, exemplo Prolog
4. Orientadas a Objeto
   - Smalltalk, Java, etc

Alternativas no projeto de linguagens

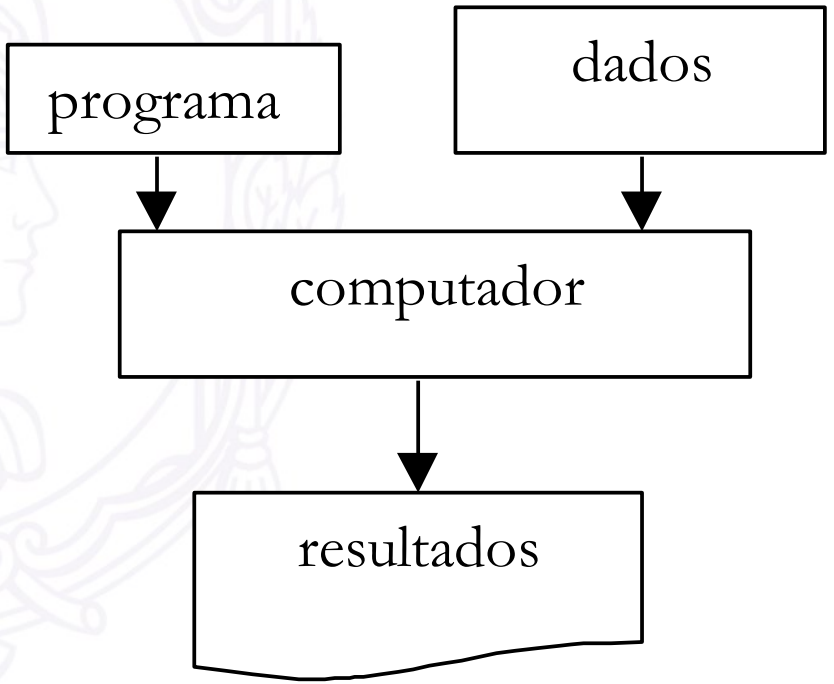
- Confiabilidade x Custo de execução
  - Verificação dinâmica dos índices de um array
- Legibilidade x Capacidade de escrita
  - APL possui vários operadores para manipular arrays, facilitando a expressão de fórmulas. A leitura, porém, é muito difícil
- Flexibilidade x Segurança
  - Aritmética de apontadores usando registros variantes (perigoso)

Métodos de Implementação

1. Compilação
   - Tradução de programas de alto-nível para código de máquina
   - Tradução lenta
   - Execução rápida
2. Interpretação pura
   - Não há tradução
   - Execução lenta
   - Têm se tornado raros
3. Sistemas de Implementação híbridos
   - Baixo custo de tradução
   - Velocidade de execução média

Execução de programas

- Componentes: **programa, dados, computador**
- Objetivo: **viabilizar a execução de um programa fonte, juntamente com seus dados, em um computador para a obtenção dos resultados**
- Problema: **notação usada no programa pode ser incompatível com o conjunto de instruções executáveis**



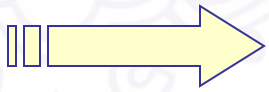
Computador e linguagens

- **Um computador pode ser representado por:**
  - **uma** máquina virtual**, capaz de executar operações mais abstratas (representadas através de uma linguagem de programação)**
  - **uma** máquina real**, capaz de executar um determinado conjunto de operações concretas (expressas em linguagem de máquina)**

L_1: máquina assembler

L_2 máquina C

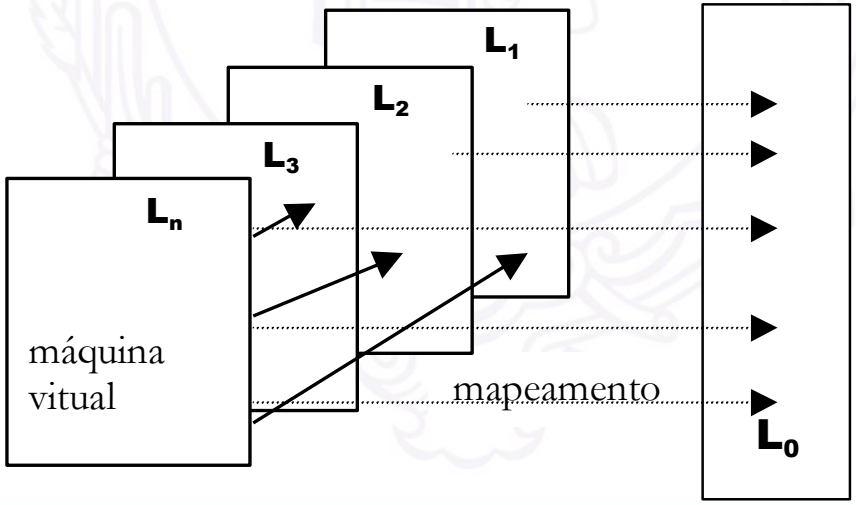
L_3 máquina Pascal



L_0: máquina real

Linguagens e Máquinas

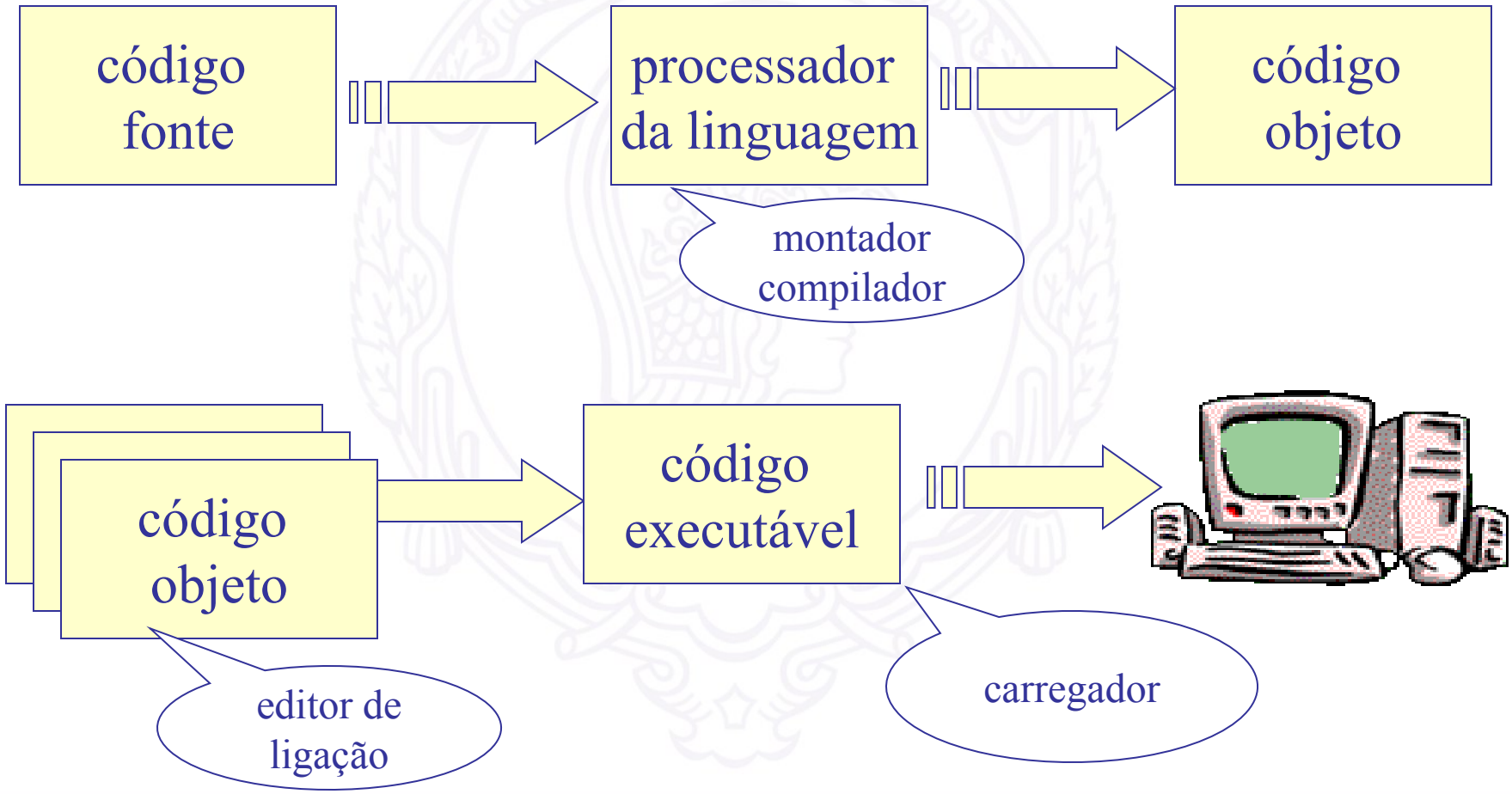
- Cada máquina representa um conjunto integrado de estruturas de dados e algoritmos
- Cada máquina é capaz de armazenar e executar programas em uma linguagem de programação L_1, L_2, L_3,...L_n.



Máquina reais e máquinas virtuais

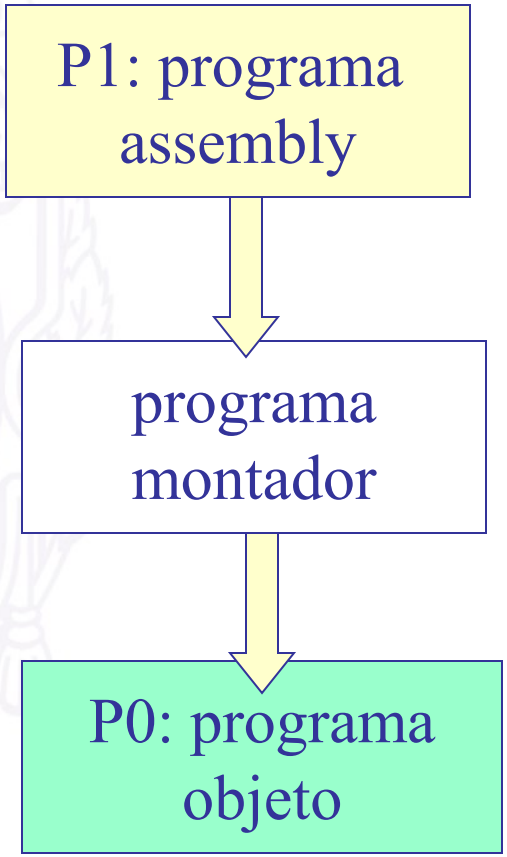
- **Máquina real:** conjunto de hardware e sistema operacional capaz de executar um conjunto próprio de instruções (plataforma de execução)
- **O elo de ligação entre a máquina abstrata e a máquina real é o processador da linguagem**
- **Processador da linguagem:** programa que traduz as ações especificadas pelo programador usando a notação própria da linguagem em uma forma executável em um computador
- **Máquina abstrata (virtual):** combinação de um computador e um processador de linguagem

Preparação de programas



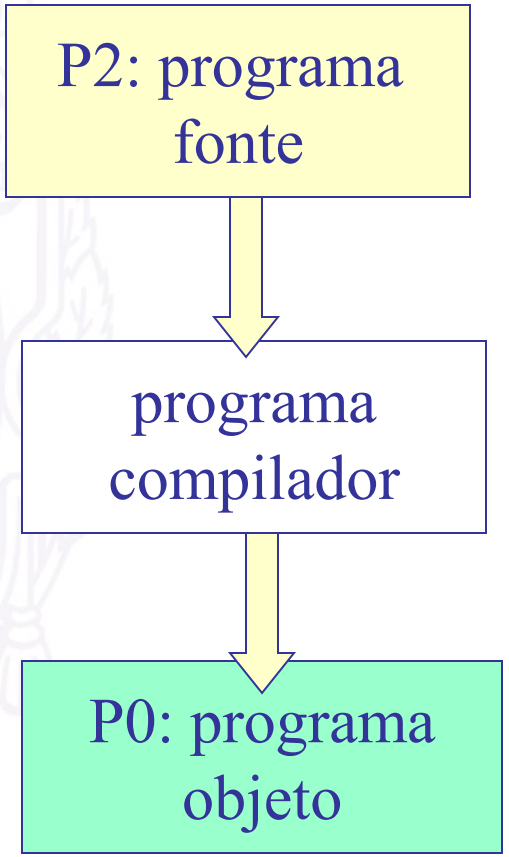
Programa Montador (*assembler*)

- **Montadores aceitam como entrada um programa escrito em linguagem de montagem (*assembly)* e produzem código de máquina correspondente a cada instrução**
- **Sendo P1 o programa em *assembly* e P0 o programa em linguagem de máquina, P1 e P0 devem ser funcionalmente equivalentes**



Programa Compilador

- Compiladores aceitam como entrada um programa escrito em linguagem de programação e produzem um programa equivalente em outro código
- Programa objeto: linguagem de máquina, linguagem assembly ou linguagem intermediária
- Sendo P2 o programa fonte e P0 o programa objeto, P2 e P0 devem ser funcionalmente equivalentes



Compiladores x plataformas

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

LTA
Laboratório de Linguagens e Tecnologias Adaptativas

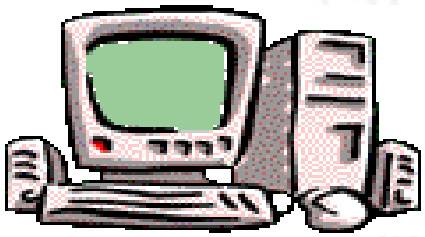
Código-fonte linguagem X

nome.X
caracteres ASCII

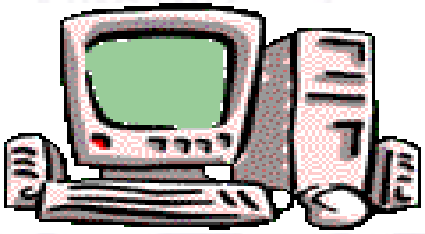
Compilador de X

Compilador de X

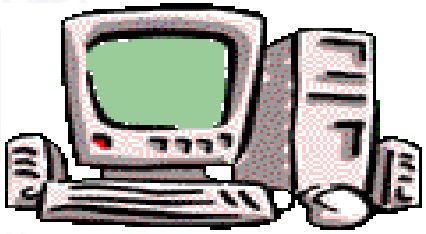
Compilador de X



PC



Mac



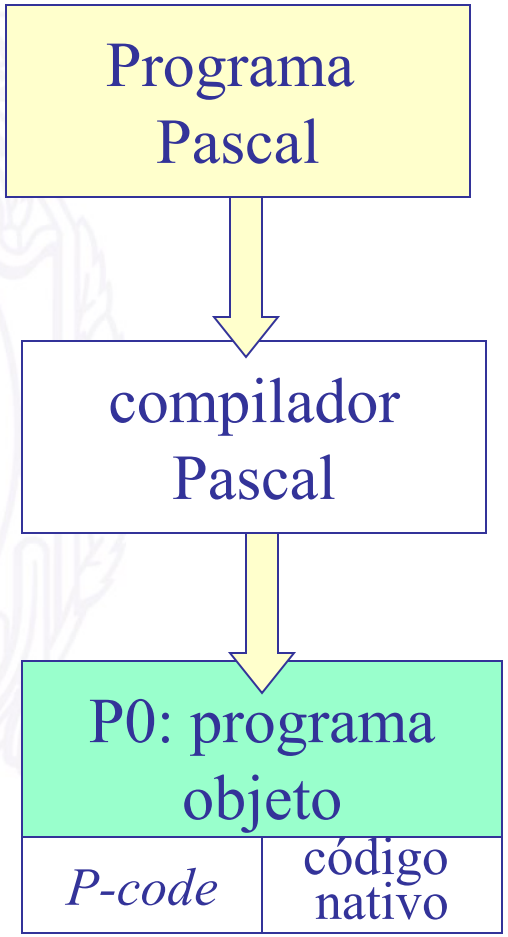
Sun

código objeto da plataforma Y (processador +SO)

nome.OBJ
nome.EXE

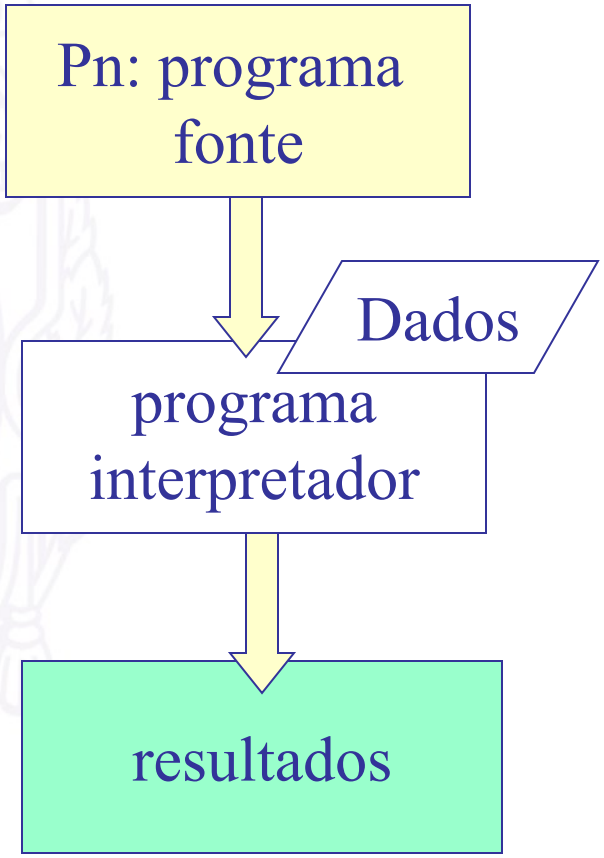
Compilador Pascal

- Programas fonte de usuários podem ser traduzidos para linguagem de máquina ou para *P-code*
- A versão em *P-code* deve ser interpretada
  - maior tempo de execução
  - melhores diagnósticos de erros
  - favorece o processo de alterações e testes
  - permite a reexecução completa ou incremental dentro do próprio ambiente de desenvolvimento



Interpretador

- Interpretador: **a partir de um programa fonte escrito em uma linguagem de programação ou *código intermediário* e mais o conjunto de dados de entrada exigidos pelo programa realiza o processo de execução**
- **Pode produzir código executável, mas não produz programa objeto persistente**
- **Exemplos: Basic, LISP, Smalltalk e Java**



Interpretadores: características

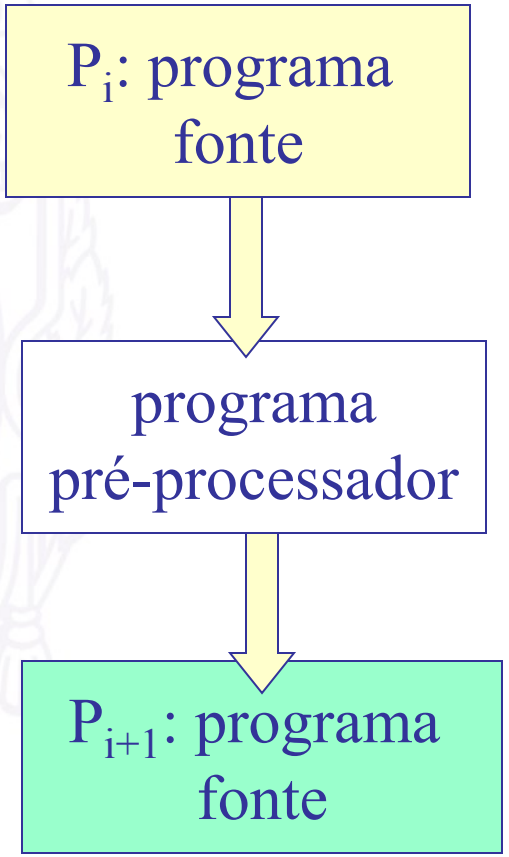
- Interpretadores se baseiam na noção de código intermediário, não diretamente executável na plataforma de destino
- **Simplicidade:** programas menores e menos complexos que compiladores
- **Portabilidade:** o mesmo código pode ser aceito como entrada em qualquer plataforma que possua um interpretador
- **Confiabilidade:** pelo fato de terem acesso direto ao código do programa (e não ao código objeto) podem fornecer ao desenvolvedor mensagens de erro mais acuradas

Compilação e execução

- Compilação:
  - geração de código executável
  - depende da plataforma de execução
  - tradução lenta X execução rápida
- Interpretação pura
  - sem geração de código
  - execução lenta, independente de plataforma
- Híbrida
  - geração de código intermediário
  - independente de plataforma de execução
  - tradução rápida X execução não muito rápida

Pré-processador

- Programa que faz conversões entre linguagens de programação de alto nível similares ou para formas padronizadas de uma mesma linguagem de programação
- Possibilita a utilização de extensões da linguagem (macros ou mesmo novas construções sintáticas) utilizando os processadores da linguagem original



Pré-processador: exemplo em C++

- Programas fonte escritos em C++ são inicialmente submetidos a um pré-processador que gera uma unidade de tradução sobre a qual o compilador irá trabalhar
- Finalidades:
  - inclusão de outros arquivos
  - definição de constantes simbólicas e macros
  - compilação condicional do código do programa
  - execução condicional de diretivas de pré-processamento

- Diretivas para o pré-processador iniciam por #
- Exemplo de macro

```
#define
CIRCLE_AREA(x) (PI * (x) * (x) )
```

- por exemplo o comando:

```
area = CIRCLE_AREA(4);
```

- será expandido para:

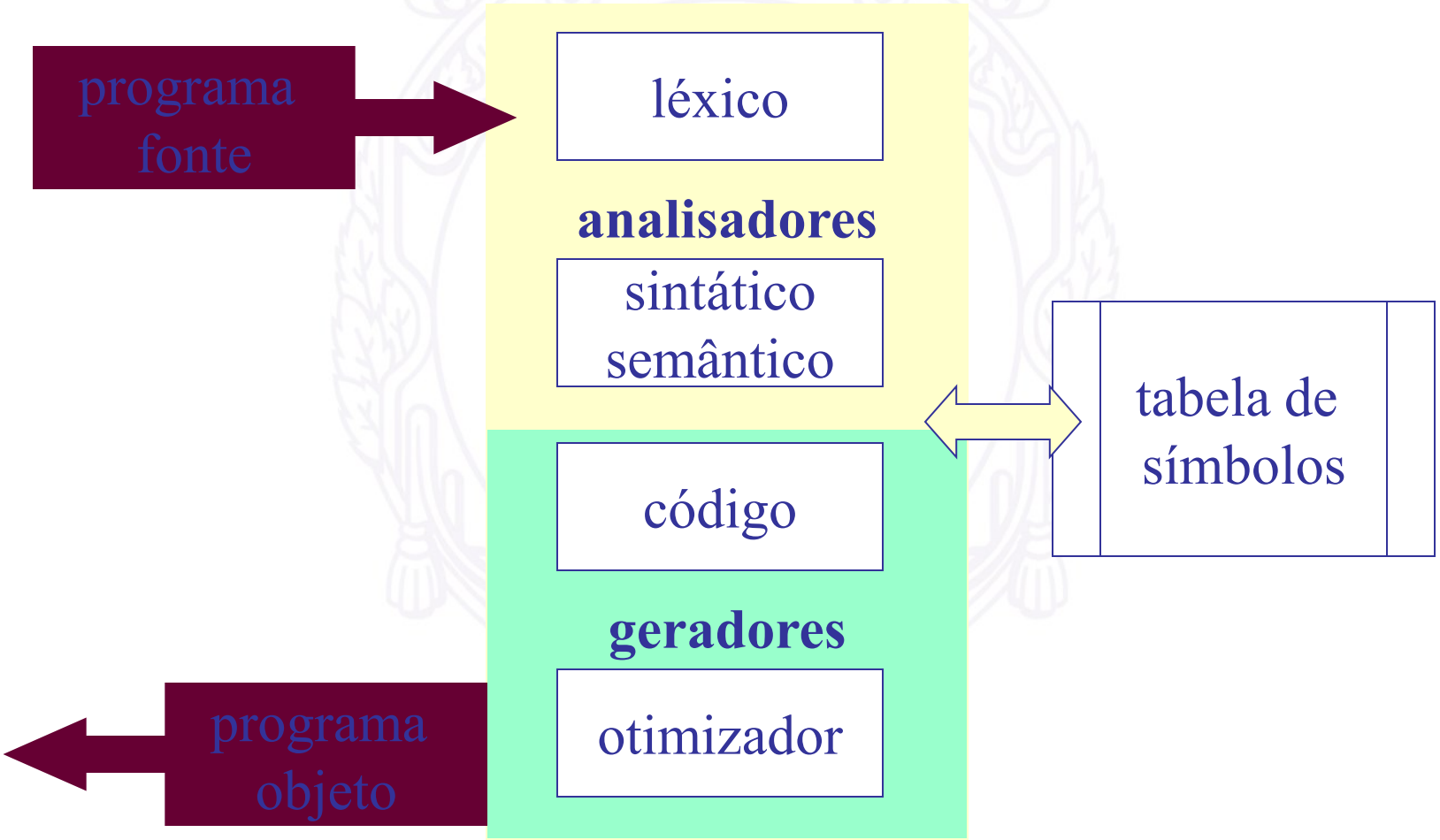
```
area=(3.14159 * (4) * (4));
```

- antes da compilação

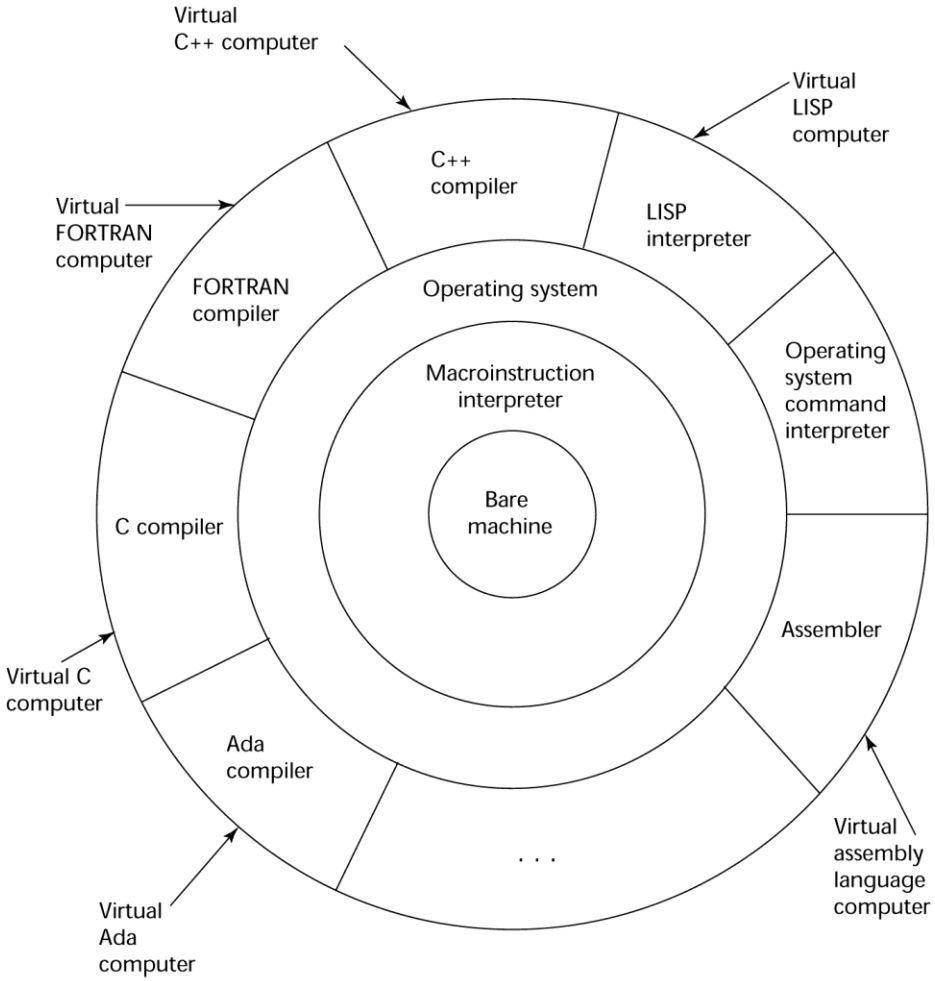
Preprocessador: exemplo em C#

```
#define Dutch

using System;

public class Preprocessor {
  public static void Main() {
    #if Dutch
      Console.WriteLine("Hallo Wereld");
    #else
      Console.WriteLine("Hello World");
    #endif
  }
}
```

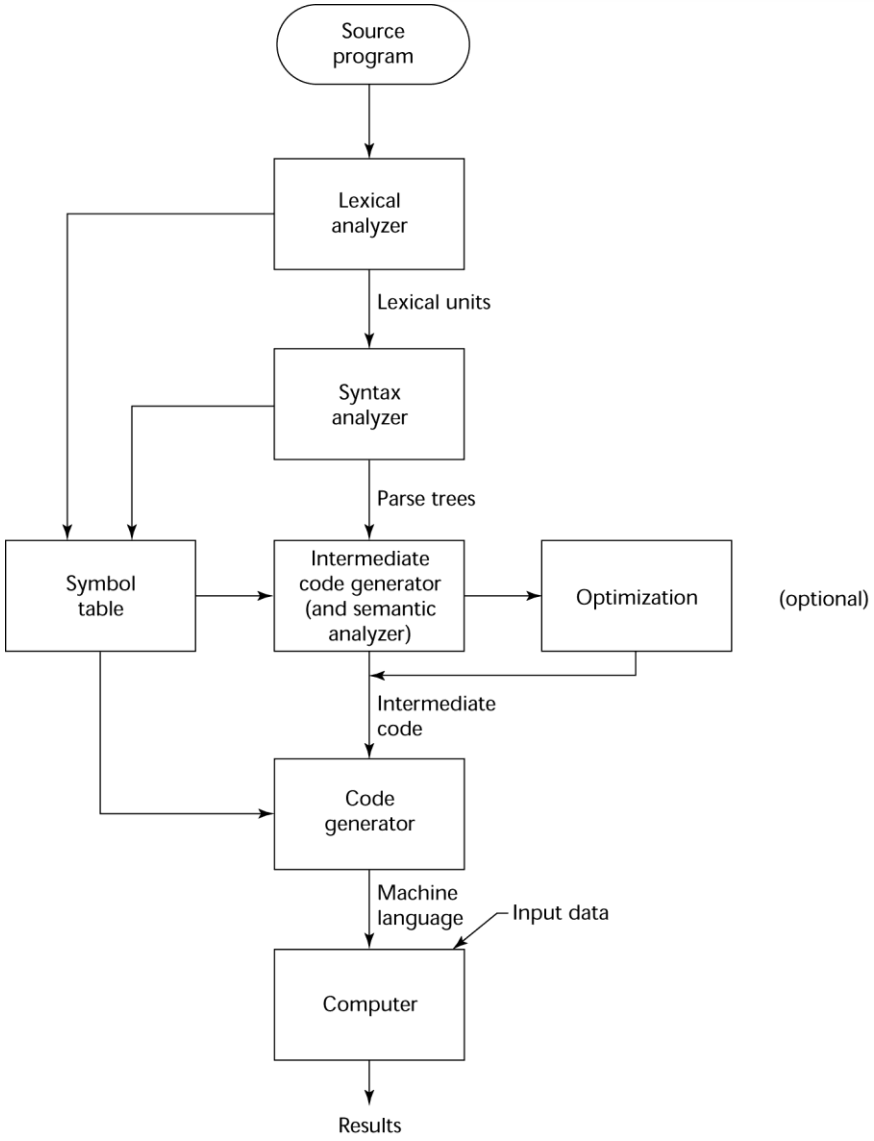
Componentes de compiladores



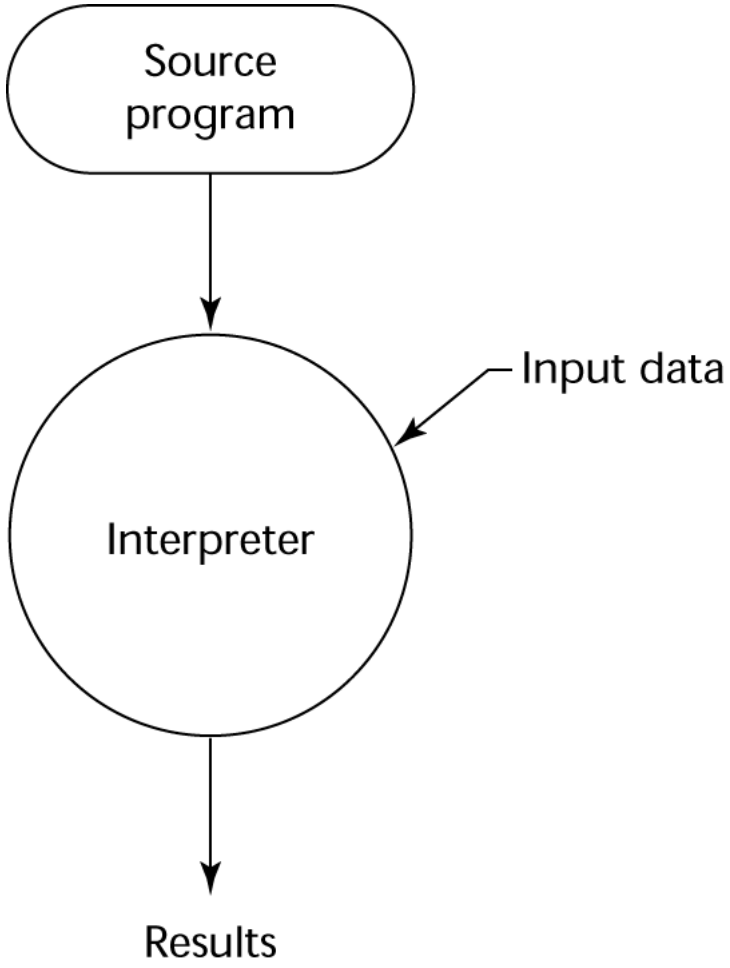
Interface de Computadores Virtuais



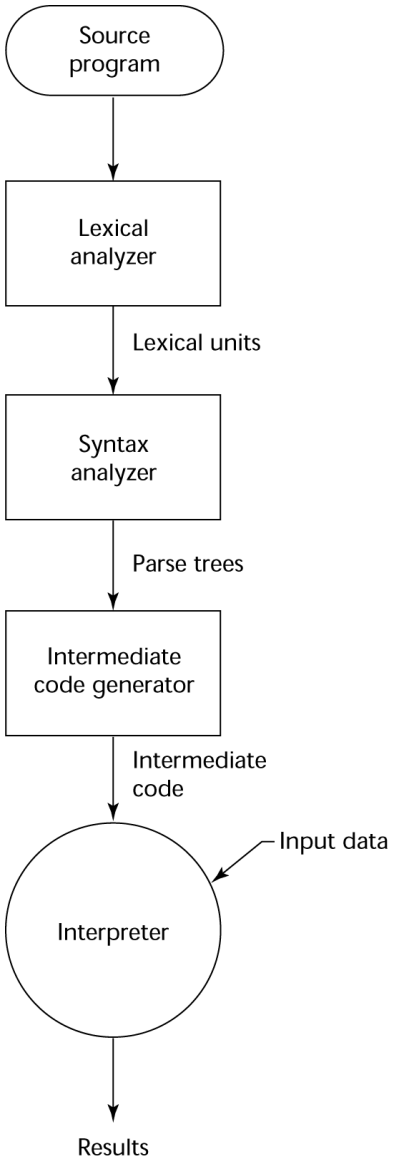
Processo de Compilação



Interpretação Pura



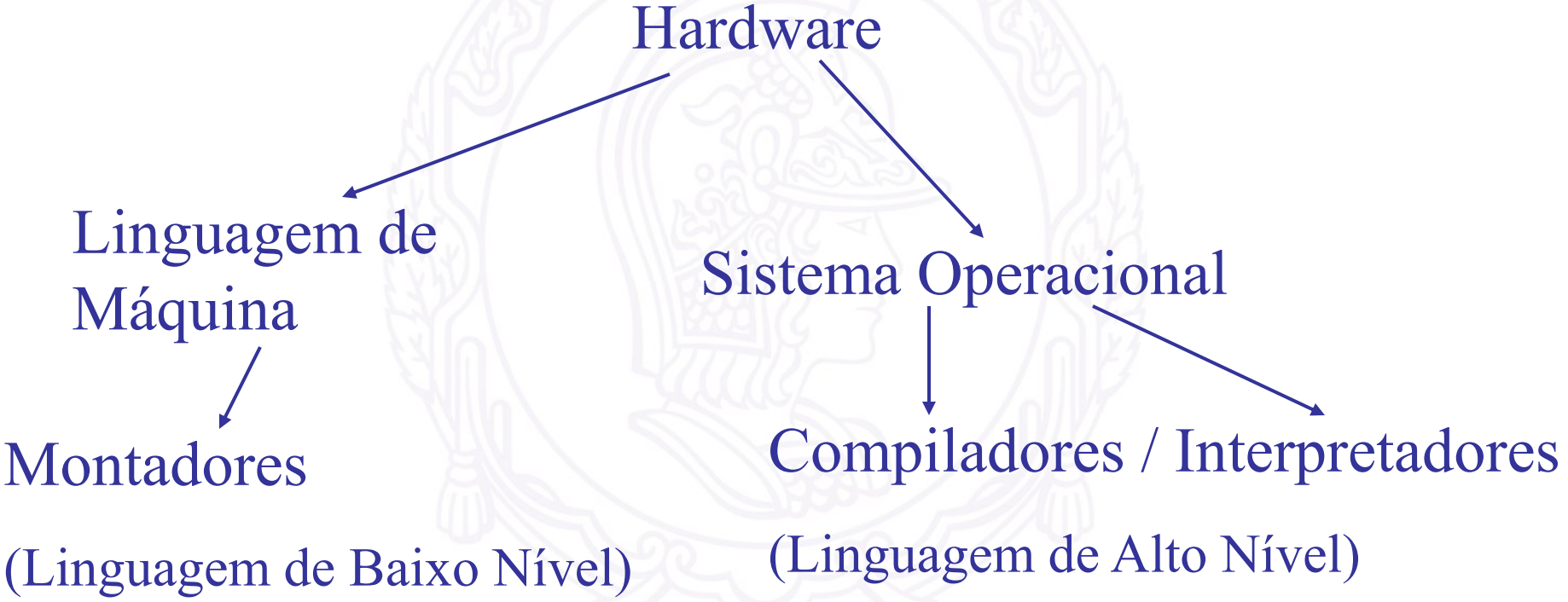
Sistema Híbrido

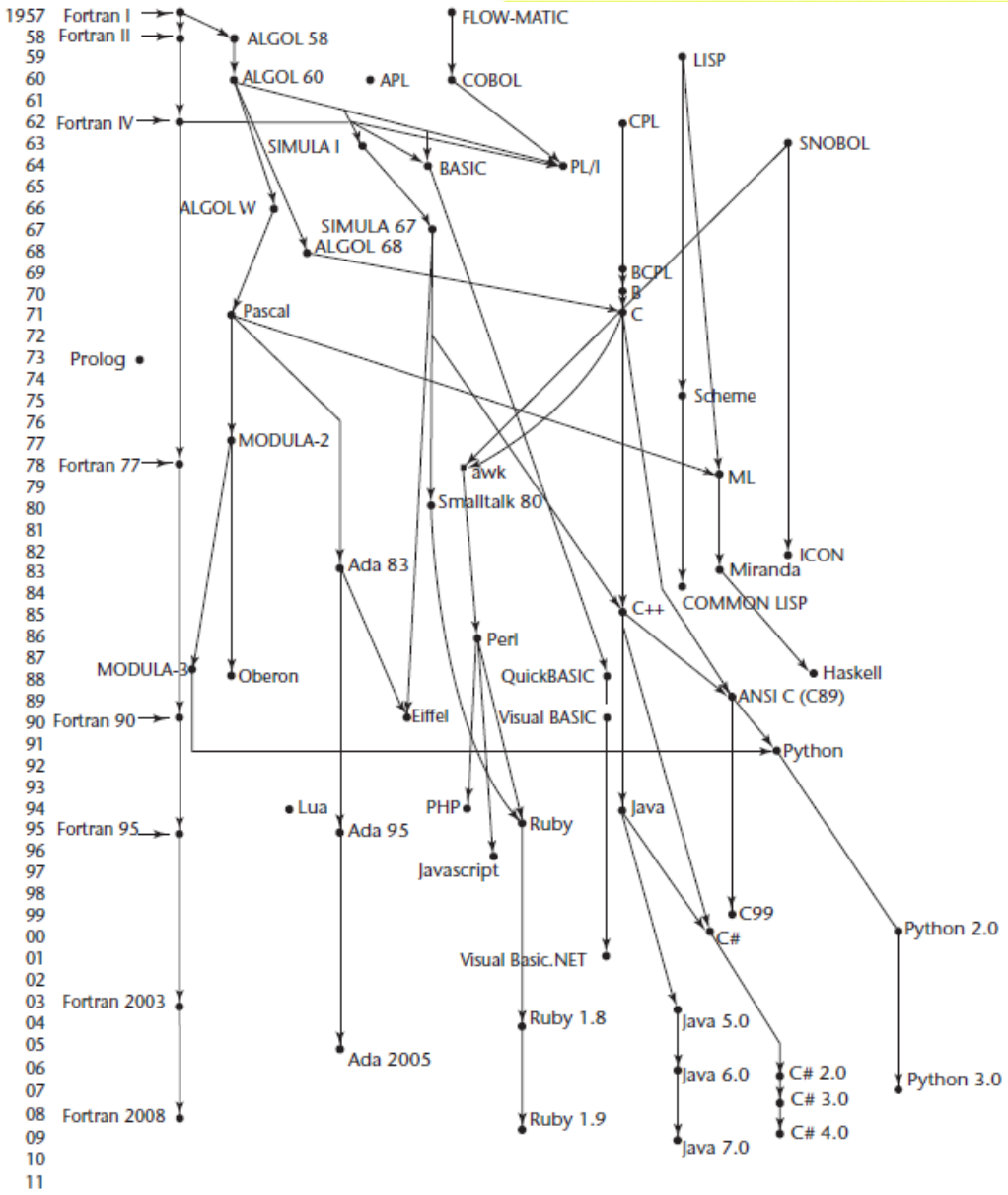


Ambientes de Programação

- Coleção de ferramentas usadas para o desenvolvimento de software, tais como:
  - UNIX
    - Um sistema operacional antigo e uma coleção de ferramentas
  - Borland C++ Builder, e Delphi
    - Ambientes de programação para o PC, voltados às linguagens C++, e Object Pascal
  - Smalltalk
    - Um ambiente/processador de linguagem
  - Microsoft Visual Studio
    - Um ambiente grande, complexo e visual

Sistema de Programação





Genealogia das Linguagens

Histórico das LPs

1. Plankalkül – 1945 (Konrad Zuse)
   - Nunca implementada (Publicação em 1972)
   - Estruturas de dados avançadas
     - Ponto flutuante, arrays, registros
   - Invariantes
   - Notação:

A(7) := 5*B(6)

| 5*B ⇒ A

V	6	7	(Subscritos)
S	1.n	1.n	(Tipos de dados)

Programação de Hardware Mínima – Pseudocódigos

2. Pseudocódigos – 1949

- O que havia de errado com o uso de código de máquina?
  - Legibilidade baixa
  - Capacidade de modificação baixa
  - Codificação de expressões tediosa
  - Deficiências de máquina – sem indexação ou ponto flutuante
- Exemplos: *Short Code* (BINAC, 1949), *Speedcoding* (IBM 701, 1954), *"Compilador"* (UNIVAC, 1951-1953).

IBM 704 e FORTRAN

3. Sistema de Laning e Zierler – 1953
   - Implementado no Whirlwind do MIT
   - Primeiro compilador "algébrico"
   - Com variáveis subscritas, chamadas de funções, tradução de expressões
   - Nunca portado para outra máquina
4. FORTRAN I – 1957 (Fortran 0 não implementado -54)
   - Projetado para o IBM 704, que possuía registradores de índice e hardware para ponto flutuante
   - Ambiente de desenvolvimento
     - Computadores eram pequenos e não confiáveis
     - As aplicações eram científicas
     - Não havia metodologia de programação ou ferramentas
     - Eficiência de máquina era o mais importante

IBM 704 e FORTRAN – cont

4. FORTRAN I – cont

- Impacto do ambiente no projeto:
  - Sem necessidade de armazenamento dinâmico
  - Necessidade de contadores e de boa manipulação de arrays
  - Sem manipulação de strings, aritmética decimal ou entrada/saída poderosa
- Características:
  - Nomes de até seis caracteres
  - Repetição com teste posterior (DO)
  - E/S formatada
  - Subprogramas definidos pelo usuário
  - Comando de seleção ternário (IF aritmético)
  - Sem declarações para tipos de dados
  - Sem compilação separada
  - Liberado em abril de 1957 após 18 H/ano de esforço
  - Programas maiores do que 400 linhas raramente compilavam principalmente devido ao IBM 704 (não confiável)
  - Código era muito rápido, tornou-se popular

IBM 704 e FORTRAN – cont

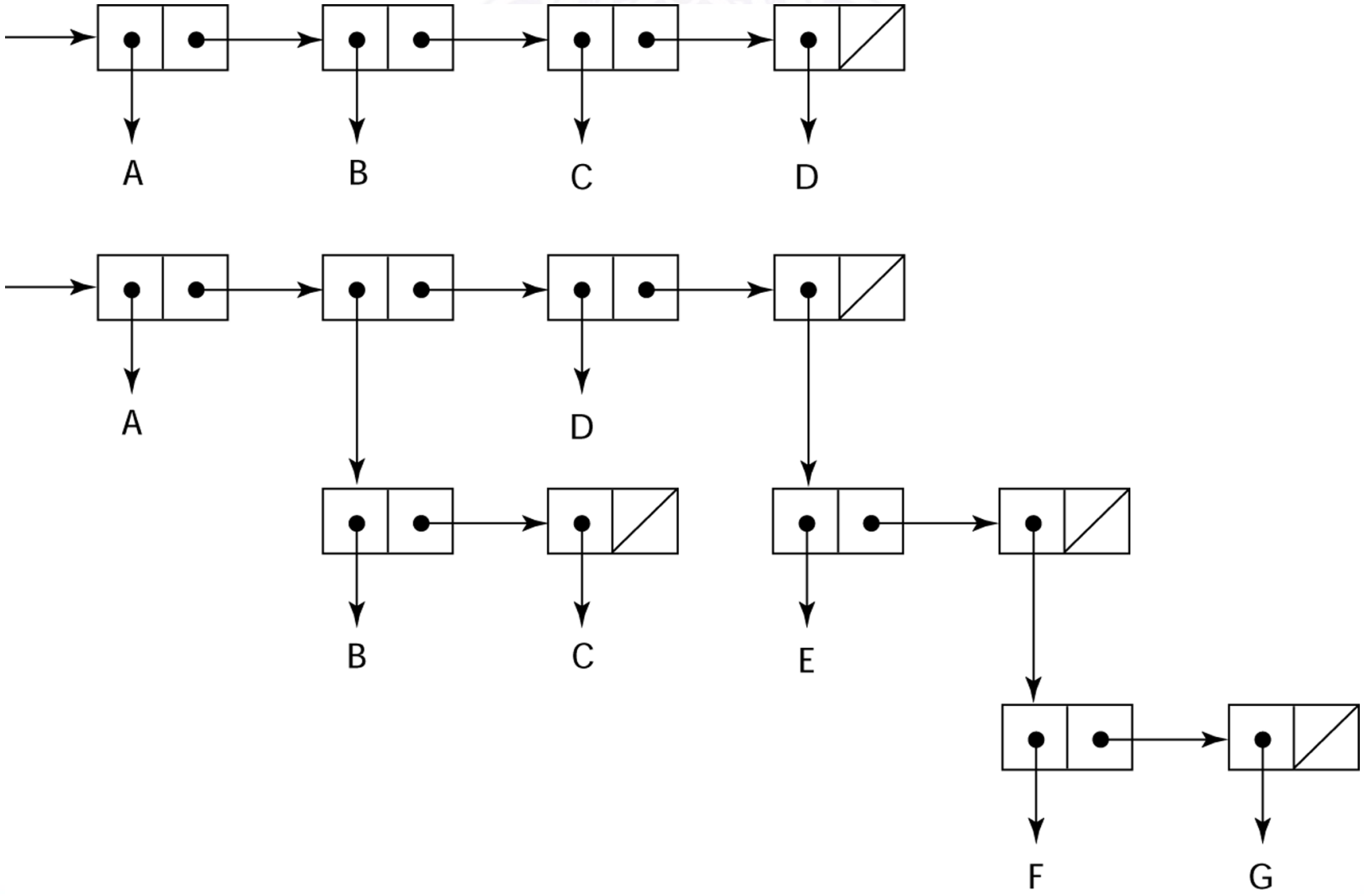
5. FORTRAN II – 1958
   - Compilação independente
   - Problemas da versão anterior corrigidos
6. FORTRAN IV – 1960-62
   - Declaração explícita de tipos
   - Comando de seleção lógica
   - Nomes de subprogramas podem ser parâmetros
   - Padrão ANSI em 1966
7. FORTRAN 77 – 1978
   - Manipulação de strings
   - Comando lógico de controle de loop
   - IF-THEN-ELSE
8. FORTRAN 90 – 1990
   - Módulos
   - Arrays dinâmicos
   - Ponteiros e recursão
   - Comando CASE e verificação de tipos de parâmetros

LISP

9. LISP (LISt Processing) – 1959

- Para processamento de listas (McCarthy no MIT)
- Pesquisa em IA necessitava de uma linguagem que:
  - Processasse dados em listas (no lugar de arrays)
  - Computação simbólica
- Só há dois tipos de dados: átomos e listas
- Sintaxe baseada em cálculo lambda
- Pioneira em programação funcional
  - Sem necessidade de variáveis ou atribuição
  - Controle por recursão ou expressões condicionais
  - Ainda é a linguagem dominante para IA
  - COMMON LISP e Scheme são dialetos contemporâneos de LISP
  - ML, Miranda, Haskell são linguagens relacionadas

Representação interna das listas



Programação Estruturada

10. ALGOL 58 – 1958

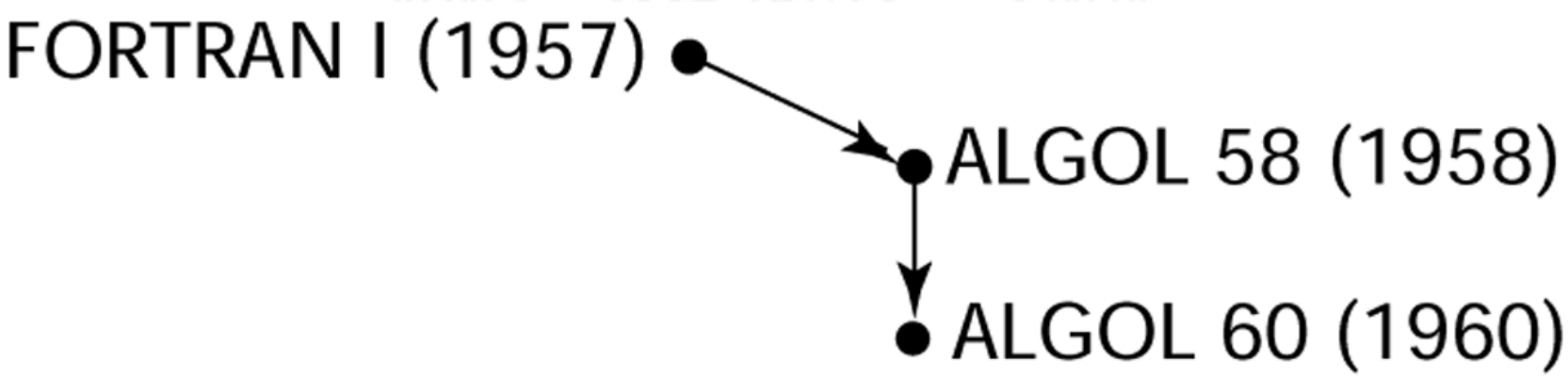
- Ambiente de desenvolvimento
  - FORTRAN mal havia aparecido para os IBM 70x
  - Várias outras linguagens em desenvolvimento para máquinas específicas
  - Nenhuma linguagem portável, todas dependentes de máquina
  - Nenhuma linguagem universal para representação de algoritmos
  - ACM e GAMM se encontram para o projeto, objetivos:
    - Proximidade da notação matemática
    - Boa para descrição de algoritmos
    - Deve ser traduzida para código de máquina
  - Características da linguagem:
    - Conceito de tipo foi formalizado
    - Nomes podem ter qualquer tamanho
    - Arrays podem ter qualquer quantidade de subscritos
    - Parâmetros separados por modo (entrada e saída)
    - Subscritos entre colchetes
    - Comandos de composição (BEGIN – END)
    - Ponto e vírgula como separador, atribuição := e IF possuía ELSEIF

ALGOL 60

11. ALGOL 60 – 1960

- Modificação sobre o ALGOL 58, novas características:
  - Estrutura de bloco (escopo local)
  - Dois métodos de passagem de parâmetros
  - Recursão de subprograma
  - Arrays dinâmicos na pilha
  - Ainda sem E/S e manipulação de strings
  - Sucessos:
    - Padrão de representação de algoritmos por mais de 20 anos
    - Todas as linguagens imperativas subseqüentes são baseadas nela
    - Primeira linguagem independente de máquina
    - Primeira linguagem cuja sintaxe foi formalmente definida (em BNF)
  - Falhas:
    - Nunca usada amplamente (especialmente nos EEUU)
    - Causas:
      - Sem E/S e seu conjunto de caracteres tornou-se não portável
      - Muito flexível (difícil de implementar)
      - Entrincheiramento do FORTRAN (pelos usuários)
      - Descrição formal da sintaxe
      - Falta de suporte da IBM

Genealogia do ALGOL 60



Linguagens Comerciais

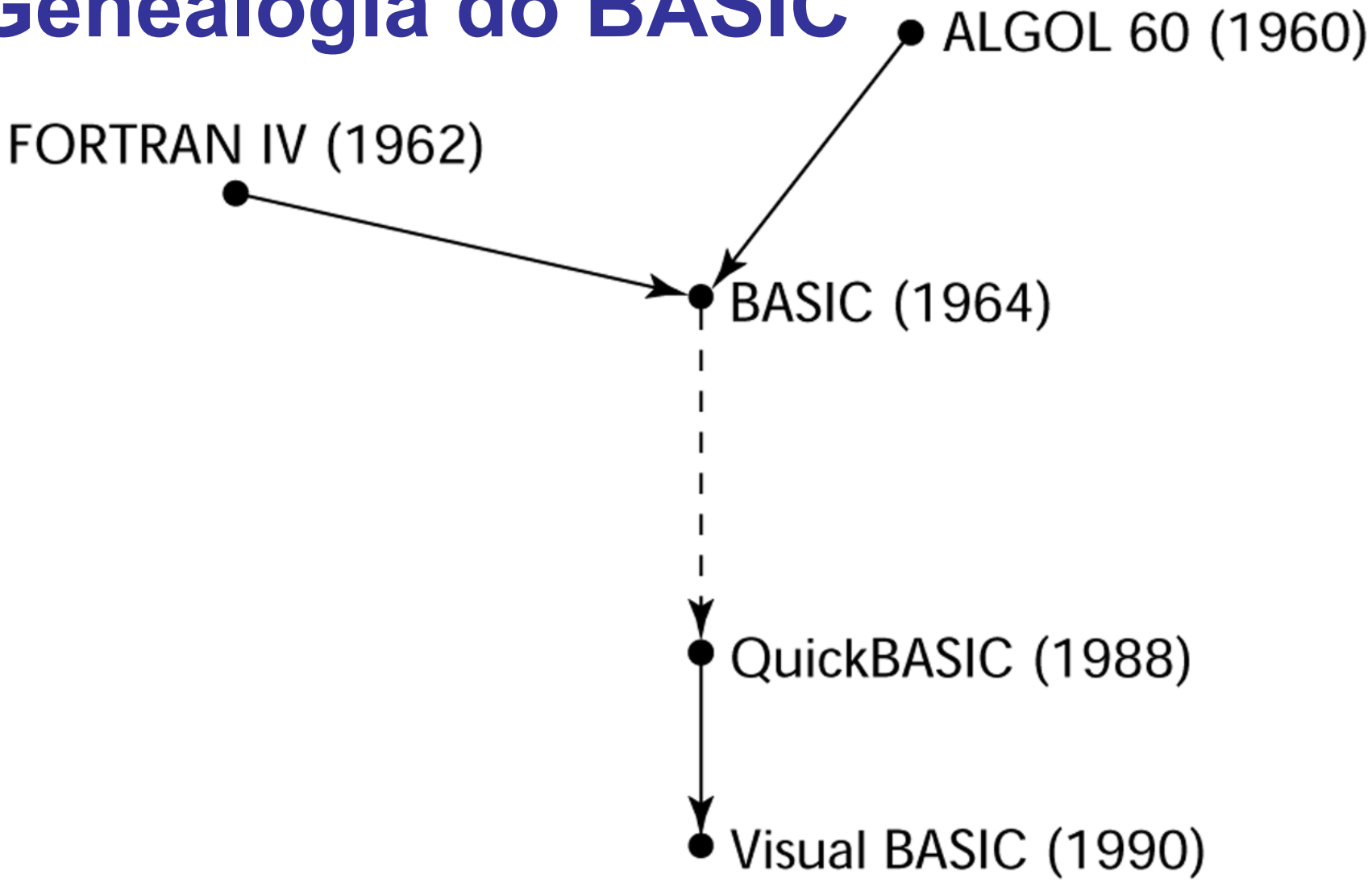
12. COBOL – 1960

- Baseada em Flow-Matic
- Objetivos do projeto:
  - Deve parecer Inglês simples
  - Deve ser fácil de usar mesmo ao custo do poder computacional
  - Deve ampliar a base de usuários de computador
  - Não deve ser restringido por problemas de implementação
  - Houve problemas no projeto (subscritos, expressões)
  - Contribuições:
    - Primeira linguagem a incluir facilidades de macros
    - Estruturas de dados hierárquicas (registros)
    - Comandos de seleção aninhados
    - Nomes longos (até 30) com hífens
    - Divisão de dados
  - Comentários:
    - Primeira linguagem requisitada pelo DoD
    - Ainda muito usada em negócios

BASIC

13. BASIC – 1964
    - Objetivos do projeto:
        - Fácil de aprender e usar para alunos de áreas não científicas
        - Deve ser “agradável e amigável”
        - Deve oferecer rápido retorno para trabalho de casa
        - Deve permitir acesso livre e privado
        - Deve considerar o tempo de usuário mais importante que o do computador
    - Dialetos populares: QuickBasic e Visual BASIC

Genealogia do BASIC

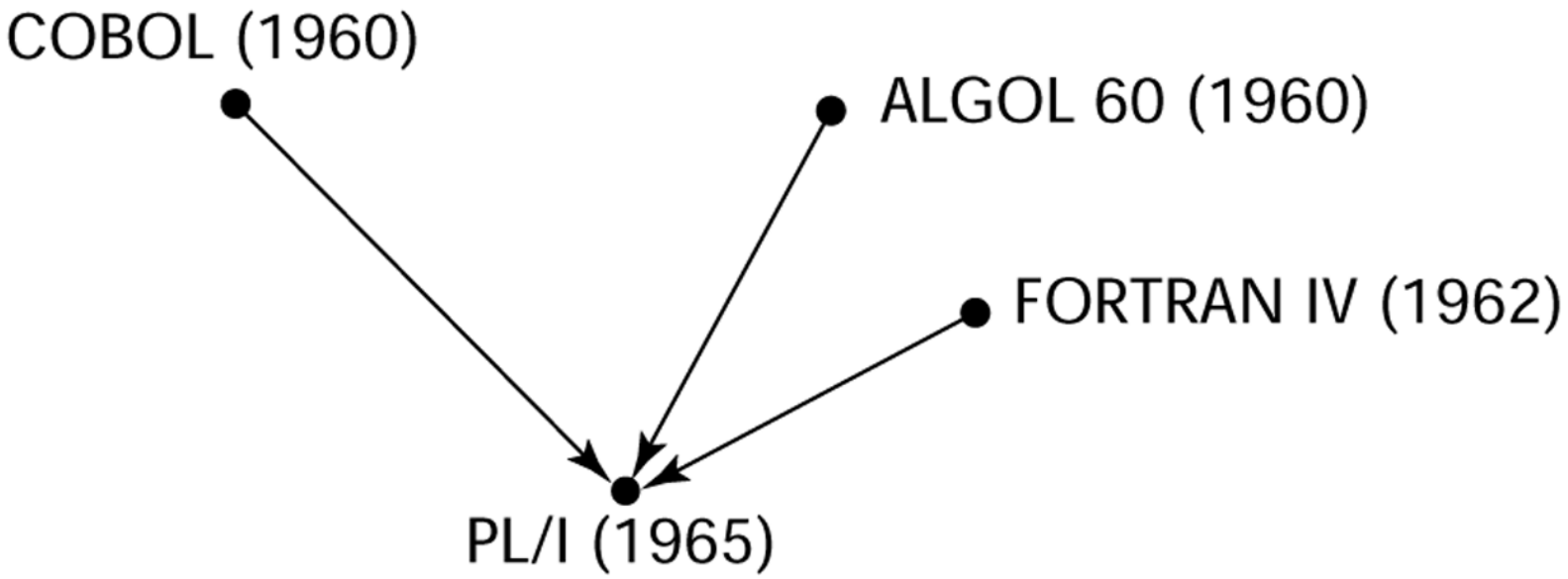


PL/I

14. PL/I – 1965

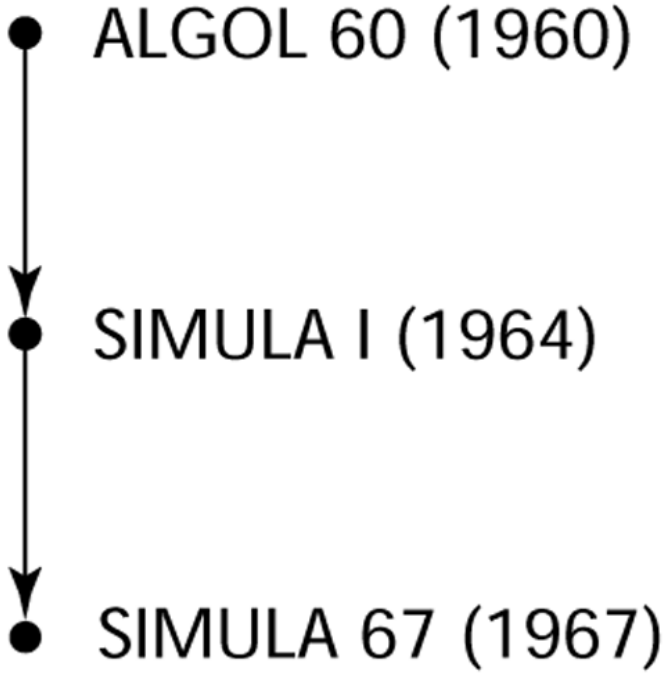
- Projetada pela IBM e SHARE
- Situação computacional à época:
  - Computação científica: FORTRAN/IBM 7090
  - Computação comercial COBOL/IBM 7080
  - Por volta de 1963 ambos os grupos começaram a precisar de características do outro (negócios de expressões, etc – MIS, científico de E/S mais elaborada)
  - Solução:
    - Construir novo computador para ambas aplicações
    - Projetar uma nova linguagem para ambas aplicações
  - Contribuições:
    - Primeira a permitir concorrência de tarefas
    - Primeiro tratamento de exceções
    - Recursão selecionável (ativável)
    - Primeiro tipo ponteiro
    - Seções transversais de arrays

Genealogia do PL/I



Linguagens Dinâmicas

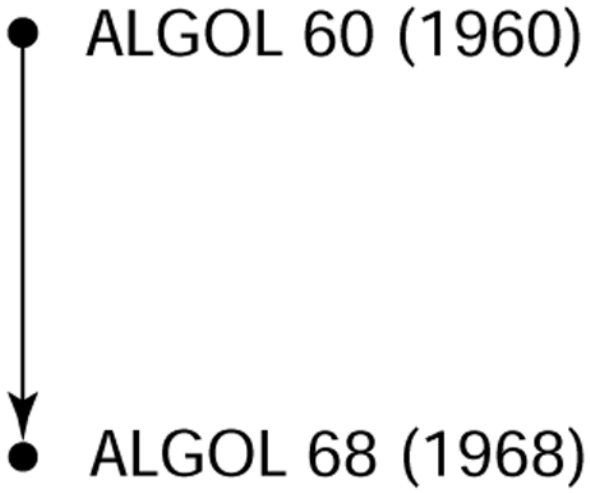
15. Primeiras Linguagens Dinâmicas
    - APL – 1962
        - Programas muito difíceis de ler
    - SNOBOL
        - Manipulação de strings
16. SIMULA 67 – 1967
    - Contribuições:
        - Co-rotinas
        - Abstração de dados (classes)



ALGOL 68

17. ALGOL 68 – 1968

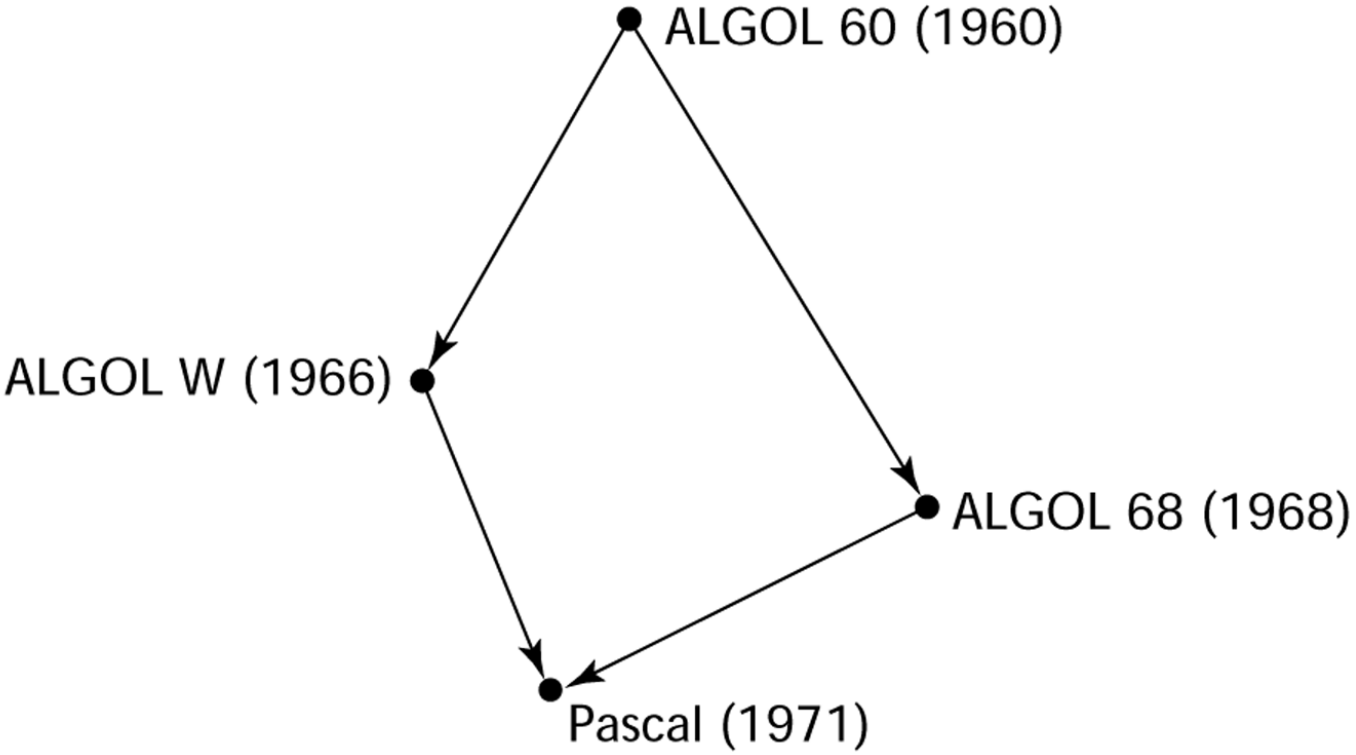
- Baseado em ortogonalidade, contribuições:
  - Estruturas de dados definidas pelo usuário
  - Tipos de referência
  - Arrays dinâmicos (chamados de *flex* arrays)
- Comentários:
  - Menos uso que ALGOL 60
  - Forte influência sobre linguagens subseqüentes (PASCAL, C, etc)



PASCAL

18. PASCAL – 1971

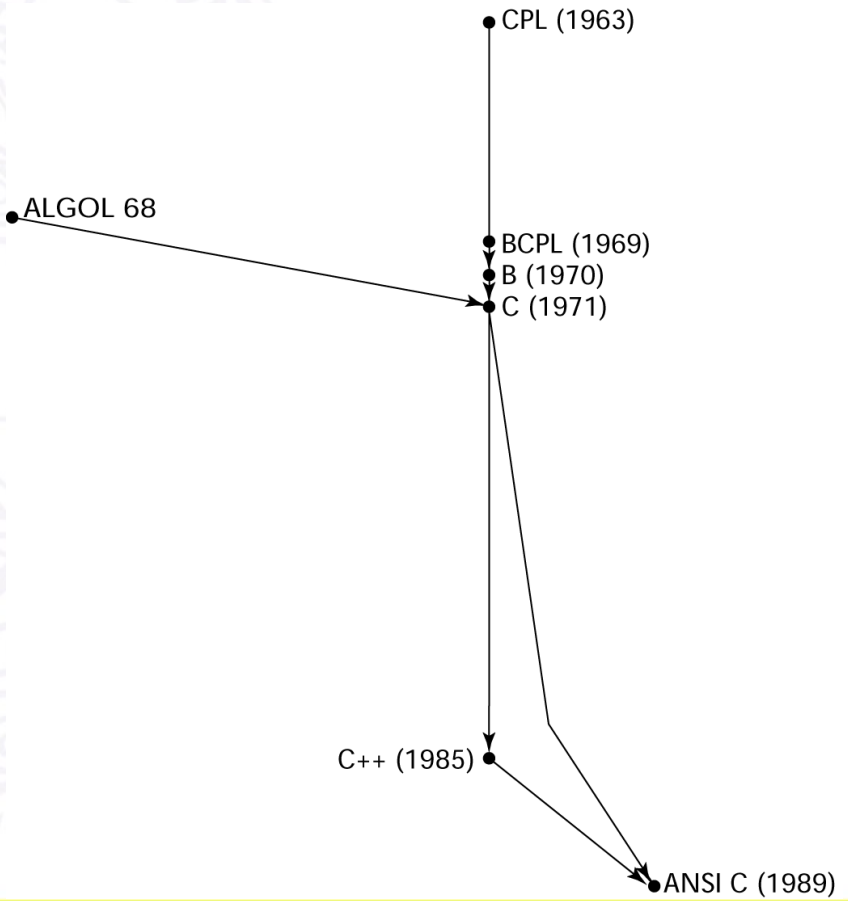
- Projetado por Wirth após deixar o ALGOL 68
- Objetivo de projeto: ensinar programação estruturada
- Pequena, simples, sem novidades
- Ainda é usada para ensino de programação (decaindo)



C

19. C – 1972

- Objetivo de projeto: Programação de sistemas
- Conjunto poderoso de operadores, e pobre verificação de tipos
- Espalhou-se por conta do UNIX



Outras

20. Outros descendentes do ALGOL

- Modula-2 (Wirth, meados dos anos 1970) – PASCAL + sistemas
- Modula-3 (Final dos 1980) – classes, concorrência, exceções
- Oberon (Wirth, final dos 1980) – Modula-2 + POO
- Delphi (Borland) – PASCAL + POO, mais elegante e segura do que C++

21. PROLOG – 1972

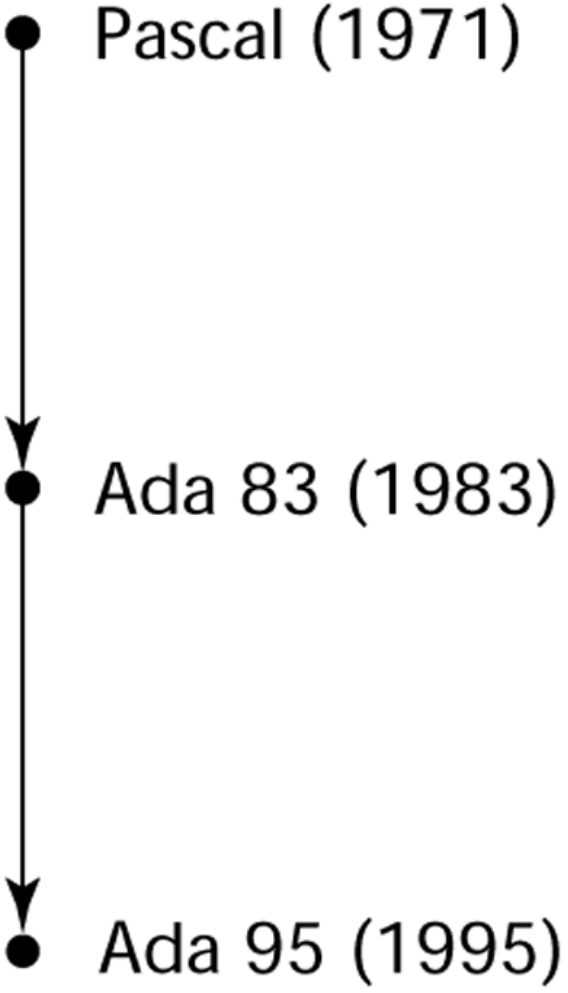
- Desenvolvida na Universidade de Marselha
- Baseada em lógica formal
- Não procedural
- Pode ser sumarizada como uma base de dados inteligente que usa um processo de inferência para inferir a verdade através de algumas cláusulas (*queries*) fornecidas

ADA

22. ADA – 1983 (começou por volta dos anos 1970)

- Contribuições:
  - Pacotes (suporte para abstração de dados)
  - Elaborado controle de exceções
  - Unidades de programa genéricas
  - Concorrência
- Comentários:
  - Projeto competitivo
  - Incluiu todos os conceitos desenvolvidos sobre Eng. SW e projeto de linguagens
  - Primeiros compiladores muito difíceis (demoraram para sair)
- ADA 95 (começou em 1988)
  - Suporte para POO (derivação de tipos)
  - Melhores mecanismos de controle para dados compartilhados (novos mecanismos de concorrência)
  - Bibliotecas mais flexíveis

Genealogia de ADA

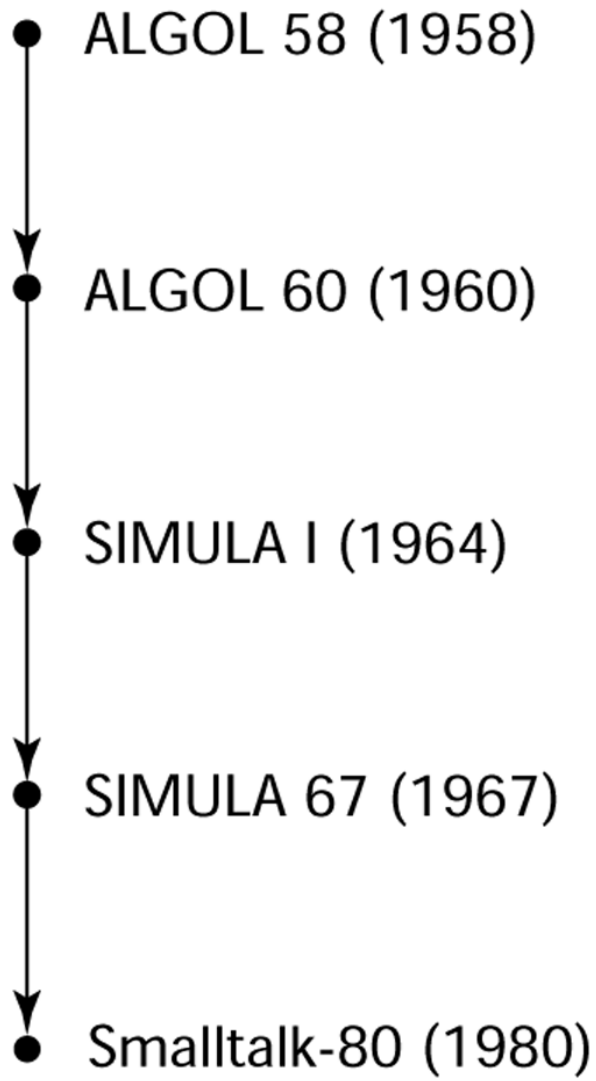


Orientação a Objetos

23. Smalltalk - 1972-1980

- Desenvolvida na Xerox PARC (Palo Alto Research Center), inicialmente por Alan Kay (baseada em sua Tese de Doutorado), posteriormente por Adele Goldberg
- Primeira implementação completa de uma linguagem orientada a objeto (abstração de dados, herança, e tipos de vincularão dinâmica)
- Pioneira na utilização de interface gráfica de usuário que todos utilizam atualmente

Genealogia de Smalltalk



C++

24. C++ – 1985

- Desenvolvida nos Laboratórios Bell por Stroustrup
- Evoluiu a partir do C e SIMULA 67
- Facilidades para programação orientada a objeto foram adicionadas à linguagem C (extraídas parcialmente de SIMULA 67)
- Também possui gerenciamento de exceções
- Uma linguagem grande e complexa, em parte porque suporta tanto programação procedural como OO
- Ganhou popularidade rapidamente, junto com OOP
- Padrão ANSI aprovado em novembro de 1997
- Eiffel - uma linguagem correlata (a C++) que suporta OOP
  - (Projetada por Bertrand Meyer - 1992)
  - Não diretamente derivada de qualquer outra linguagem
  - Menor e mais simples que C++, mas retém a maior parte da potencialidade
- Go (pouco relacionada)

Java

25. Java (1995)

- Desenvolvida na Sun no início dos anos 1990
- Baseada em C++
- Significativamente simplificada
- Suporta *somente* OOP
- Tem referências, mas não possui ponteiros
- Inclui suporte a *applets* e uma forma de concorrência

Popularidade TIOBE (Jul/2014)