



# **INTRODUÇÃO À SISTEMAS DE INFORMAÇÃO**

**AULAS 18: LINGUAGEM DE  
PROGRAMAÇÃO**

**PROF<sup>a</sup>: LEONARA BRAZ  
LEONARABRAZ@GMAIL.COM**



# SEÇÃO 1

## LINGUAGENS DE PROGRAMAÇÃO

# INTRODUÇÃO

- Para a implementação de um algoritmo é necessário descrevê-lo de maneira que ele seja apto a ser executado pelo computador.
- O fator de sucesso ou fracasso de uma aplicação deve-se à escolha de uma **Linguagem de Programação** adequada para o seu desenvolvimento.

# INTRODUÇÃO

- As Linguagens de Programação vêm passando por um processo constante de evolução desde as últimas quatro décadas.
- Diversas linguagens foram aparecendo, obrigando-nos a avalia-las segundo alguns critérios que as diferenciavam por suas características

# DEFINIÇÃO

Linguagem de programação é um **conjunto de regras sintáticas e semânticas** usadas para **definir o algoritmo** de um software e **dizer** ao computador **o que deve ser feito**

# DEFINIÇÃO

- Uma linguagem de programação é usada para escrever o **código-fonte** de um software
- O código-fonte de um software é a **representação** do seu algoritmo em uma linguagem de programação
- Código-fonte consiste de **uma sequência de instruções**



# SEÇÃO 2

## HISTÓRICO

# HISTÓRICO – DÉCADA DE 40

- 1ª geração de linguagens
  - Softwares eram escritos em código específico para o hardware formados principalmente por números.
  - As chamadas linguagem de máquina



# HISTÓRICO – DÉCADA DE 40

- 2ª geração de linguagens
  - Simplificação do processo de programação com uso de instruções formadas por palavras mnemônicas
  - O conjunto de palavras mnemônicas é a linguagem Assembly

# HISTÓRICO – DÉCADA DE 40

- **Exemplo:** Mover o conteúdo do registro 5 para o registro 7

- Código usado nos primeiros softwares

4057

- Código usado em Assembly

MOV R5, R7

# HISTÓRICO – DÉCADA DE 40

- Assembly iniciou o uso de **palavras mnemônicas** para representar endereços de memória (variáveis)
- Softwares chamados **assemblers** realizavam a conversão do código-fonte em Assembly para a linguagem de máquina
- Assembly era **dependente da máquina** em que o software executaria, ou seja, se o software era executado em uma máquina diferente daquela que foi escrito originalmente, eram necessários ajustes

# HISTÓRICO – DÉCADA DE 40

- Assembly necessita que o programador pense **em passos bem pequenos e simples no desenvolvimento**, tornando o desenvolvimento de software mais difícil
- Tarefas simples necessitavam de **muitas instruções** em Assembly

# HISTÓRICO – DÉCADA DE 50

- 3ª geração de linguagens de programação
  - Necessidade de uma linguagem de programação que seria **independente de máquina**, ou seja, um mesmo código-fonte geraria um software que funcionaria em qualquer computador
  - Exemplos:
    - FORTRAN (FORmula TRANslator)
    - COBOL (COmmon Bussiness-Oriented Language)
    - LISP (LISt Processor)

# HISTÓRICO – DÉCADA DE 50

- Possibilita a **portabilidade** da linguagem de programação
  - Portabilidade é a execução de um software com nenhuma ou poucas modificações no código-fonte em computadores com arquitetura e/ou Sistemas Operacionais diferentes

# HISTÓRICO – DÉCADA DE 50

- Linguagens formadas por várias primitivas:
  - Variáveis, palavras reservadas, estruturas de seleção, etc
  - **Exemplo:** calcular o valor total da compra, sendo que o valor é formado pelo preço do produto mais o valor do imposto.

$\text{Gasto} = \text{preçoProduto} + \text{imposto}$

# HISTÓRICO – DÉCADA DE 50

- **Compilador** é um software que **traduz** o código-fonte de uma linguagem de programação em um arquivo executável
- A tradução realizada por um compilador é chamada de compilação
- O arquivo executável é composto por **instruções em linguagem de máquina** da arquitetura em que o software executará
  - Exemplo de arquiteturas: ARM, Intel x86, etc



# HISTÓRICO – DÉCADA DE 50

- A compilação além da arquitetura depende do SO onde o programa executará
- Dois Sistemas Operacionais diferentes podem ter diferentes comandos para abrir um arquivo
- Um mesmo compilador possui uma versão para cada SO suportado por compilador

# HISTÓRICO – DÉCADAS DE 60 E 70

- Este período trouxe um grande florescimento de linguagens de programação.
  - A maioria dos principais paradigmas de linguagem agora em uso foram inventados durante este período
- Iniciou-se o debate sobre programação estruturada
  - Este paradigma tem ênfase no uso de sub-rotinas, laços de repetição, condicionais e estruturas em bloco.
- Exemplos de Linguagens deste período:
  - Pascal, C, Prolog

# HISTÓRICO – DÉCADAS DE 80

- Linguagens Orientadas a Objeto
- Surgimento de linguagens lógicas e funcionais
  - **Exemplo:**
    - Prolog e Haskell
- Expansão das linguagens existentes e adaptação aos novos contextos

# HISTÓRICO – DÉCADAS DE 90

- Era da maturação das ideias antigas
  - Busca da reestruturação e recombinação da linguagem
- Busca da produtividade do programador
- Surgimento de linguagens com "aplicações de desenvolvimento rápido" (RAD)
  - **Exemplo:** Object Pascal, Visual Basic, e C#

# HISTÓRICO – DÉCADAS DE 90

- Surgimento das novas linguagens de Scripting
  - Contaram com sintaxes novas e incorporação mais liberal de novas funcionalidades
  - Facilidade para escrever e manter pequenos programas
- Passaram a ser mais utilizadas em conexão com a web
  - **Exemplo:** JavaScript, PHP, Python

# HISTÓRICO – TENDÊNCIAS ATUAIS

- A evolução das linguagens de programação continua, tanto na indústria quanto na pesquisa. Algumas das tendências atuais incluem:
  - Mecanismos para a adição de segurança e verificação da confiabilidade para a linguagem
  - Maior ênfase na distribuição e mobilidade
  - Suporte para Unicode de forma que o código-fonte não esteja restrito aos caracteres contidos no código ASCII
  - XML para a interfaces gráficas



# SEÇÃO 2

## PARADIGMAS DE PROGRAMAÇÃO

# PARADIGMAS DE PROGRAMAÇÃO

- As gerações de linguagens de programação podem ser usadas para mostrar a mudança do grau de proximidade da linguagem com comandos de máquinas ou comandos pensados por humanos



# PARADIGMAS DE PROGRAMAÇÃO

- As linguagens de programação não evoluíram exatamente no sentido da utilização de comandos ou linguagem dos humanos na programação
- As linguagens de programação evoluíram de maneiras diferentes seguindo alguns paradigmas

# PARADIGMAS DE PROGRAMAÇÃO

- As Linguagens de Programação podem então, serem divididas de acordo com o seguinte critério:
  - Linguagens de baixo nível
  - Linguagens não estruturais
  - Linguagens procedurais
  - Linguagens lógicas
  - Linguagens funcionais
  - Linguagens orientadas a objeto

# LINGUAGENS DE BAIXO NÍVEL

- São linguagens cujas instruções estão mais próximas ou correspondem quase diretamente ao código de máquina.
- As linguagens de máquina estão associadas ao hardware/processador que o código seria rodado

# LINGUAGENS NÃO ESTRUTURADAS

- Linguagens que não estão vinculadas ao processador utilizado
- Utilização de semântica mais genérica
  - Podendo ser utilizada em diferentes plataformas sem alteração

# LINGUAGENS PROCEDURAIS

- Define o que o software fará através de uma sequência de instruções que cria e manipula dados
- Programação procedural elimina uso do comando **goto** com utilização de estruturas de seleção, repetição e funções
- Programação procedural agilizou e facilitou o desenvolvimento de software

# LINGUAGENS PROCEDURAIS

- As linguagens procedurais são consideradas subtipos das linguagens estruturadas.
  - Possuem uma estrutura de controle, organizando de forma mais eficiente e clara a sintaxe de um programa
- Exemplo:
  - Estrutura de teste
    - If, THEN, else
  - Estrutura de repetição de blocos
    - FOR

# LINGUAGENS LÓGICAS

- Programação lógica é um paradigma de programação que faz uso da lógica matemática
  - Utilização de linguagens lógicas
- Usa estruturas pré-estabelecidas para definir o problema
- Muito usado em sistemas de simulação para testar hipóteses e situações

# LINGUAGENS FUNCIONAIS

- A linguagem funcional trata a computação como uma avaliação de **funções matemáticas** e que evita estados ou dados mutáveis.
  - Software é visto como uma função que aceita entradas e funções de saída
  - Código-fonte é composto de várias funções que combinadas produzem o resultado esperado
- Este tipo de programação está mais voltado à avaliação de expressões formadas com a utilização de funções que procuram combinar valores básicos e não simplesmente executar comandos como as linguagens estruturais.



# LINGUAGENS ORIENTADAS A OBJETO

- Surgiram em razão da necessidade gerada pelas novas técnicas de análise, apontada pelos estudos da Engenharia de Software, com a finalidade de orientar e organizar os processos
- A visão, até então estruturada (top-down) passou a ser bottom-up

# LINGUAGENS ORIENTADAS A OBJETO

- Software é desenvolvido usando um conjunto de objetos que representam elementos do mundo real necessários para realização das tarefas do programa
- Os objetos possuem atributos e métodos e são metáforas de objetos do mundo real
  - Atributos contém as características de um objeto
  - Métodos são as ações que um objeto pode realizar



# SEÇÃO 3

## EXEMPLO DE LINGUAGENS

# ASSEMBLY

- Surgiu em meados da década de 50, iniciando a Segunda geração de linguagens de programação.
- Invenção dos transistores e conseqüentemente a evolução dos computadores, veio a necessidade de práticas programáticas mais simplificadas e ágeis, porém mantendo o proximidade com o nível de máquina.

# ASSEMBLY

- Usada para escrever comandos em linguagem de máquina
  - Utilizada para programar códigos entendidos por dispositivos computacionais, como microprocessadores e micro controladores
- O código de máquina torna-se legível pela substituição dos valores em bruto por símbolos chamados mnemónicos
  - **Exemplo:** enquanto um computador sabe o que a instrução-máquina IA-32 (B0 61) faz, para os programadores é mais fácil recordar a representação equivalente em instruções mnemónicas `MOV AL, 61h`

# ASSEMBLY

- A tradução do código Assembly para o código de máquina é feita pelo montador ou **assembler**.
  - Por isso é frequentemente chamado de linguagem de montagem
- Ele converte os mnemônicos em seus respectivos opcodes, calcula os endereços de referências de memória e faz algumas outras operações para gerar o código de máquina que será executado pelo computador.

# ASSEMBLY

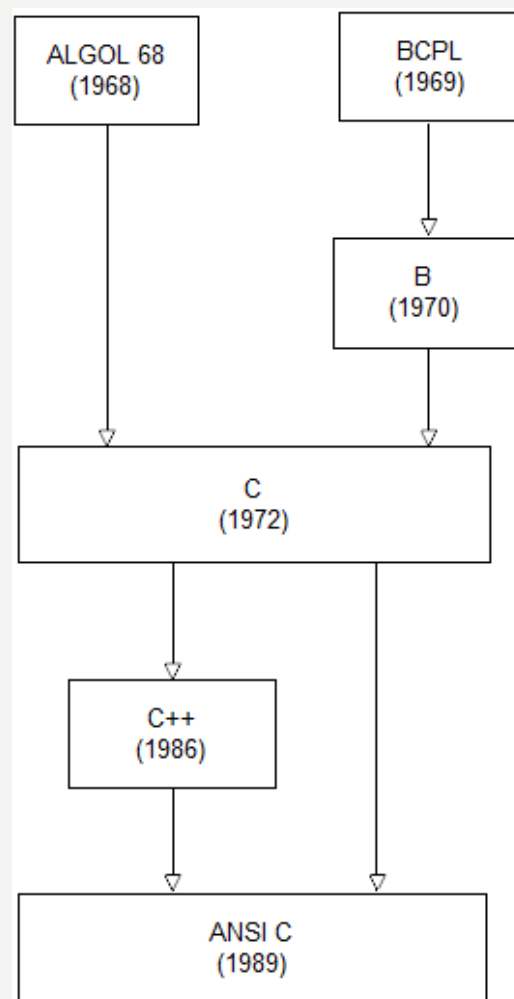
- **VANTAGEM:**

- Assembly é rápido
- Acesso direto ao hardware
- Segurança
- serve para entender como as coisas funcionam

- **DESVANTAGENS:**

- Complexidade alta
- Baixa produtividade do programador Assembly.
- Linguagem crua e com poucos "recursos".
- Difícil manutenção do código em Assembly.

# C





# C

- C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural
- C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para ela
  - Portabilidade possibilitando que um código-fonte com poucas ou nenhuma modificação crie executáveis para diversas plataformas

# C

- Difícil detecção de alguns erros
- Usada no desenvolvimento do UNIX
- A linguagem C foi criada com um objetivo principal em mente:
  - Facilitar a criação de programas extensos com **menos erros**, recorrendo ao paradigma da programação procedural, mas sobrecarregando menos o autor do compilador

# C++

- Desenvolvida por Bjarne Stroustrup em 1983 como um adicional à linguagem C
- C++ é desenvolvido para ser uma linguagem de propósito geral que é tão eficiente e portátil quanto o C.
- Desenvolvido para ser o mais compatível com C possível, fornecendo transições simples para código C.
- Desenvolvida para suportar múltiplos paradigmas.
- Muito utilizada no desenvolvimento de software

# C++ - CARACTERÍSTICAS

- Programação orientada a objeto
- Compatibilidade com C
- Programação Modular
- Portabilidade

# C#

- Linguagem de programação orientada a objetos
  - Faz parte da plataforma .NET.
- Baseada na linguagem C++
- Programação orientada a objetos
  - Utilização de classe e subclasses, associação e abstração

# C#

- Características Principais da Linguagem
  - Clareza, simplicidade e facilidade
  - Completamente orientada a objetos
  - Fortemente tipada
  - Não requer ponteiros para gerenciar a memória

# COBOL

- É uma linguagem de programação orientada para o processamento de banco de dados comerciais
- Sua meta é servir como uma linguagem de programação para negócios. Os programas para negócios não precisam de cálculos tão precisos como os encontrados em engenharia, assim o COBOL foi concebido basicamente com as características:
  - Acesso rápido a arquivos e bases de dados
  - Atualização rápida de arquivos e bases de dados
  - Geração de uma grande quantidade de informações
  - Saída com um formato compreensível ao usuário

# JAVA

- Java é uma linguagem de programação interpretada orientada a objetos
- A linguagem Java foi projetada tendo em vista os seguintes objetivos:
  - Orientação a objetos
  - Portabilidade - Independência de plataforma - "escreva uma vez, execute em qualquer lugar"
  - Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP
  - Segurança - Pode executar programas via rede com restrições de execução



# JAVA

- Muito utilizada em sistemas embarcados e aplicações web (Applets ou JSF)
- Muitas bibliotecas e frameworks
- Gerenciamento da memória por parte da máquina virtual
- Pode ser utilizada como o paradigma funcional
- Linguagem padrão para desenvolvimento de aplicativos para Android

# JAVASCRIPT

- Foi originalmente implementada como parte dos navegadores web para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor
- É uma linguagem imperativa e orientada a objetos
- Tipagem dinâmica, ou seja, o tipo da variável é identificado pelo interpretador em tempo de execução
- Muito utilizada em aplicações web

# JAVASCRIPT

- Os programas são arquitetados na forma de objetos que interagem entre si
- Cada classe determina o comportamento (definidos nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

# JAVASCRIPT - CARACTERÍSTICAS

- **Client-Side**

- Linguagem que roda na máquina cliente. O próprio navegador do cliente é o encarregado de interpretar as instruções JavaScript e executá-las para realizar as operações programadas.

- **Interpretada**

- JavaScript não precisa ser compilada, pois é uma linguagem interpretada diretamente Pelo navegador.

- **Case-Sensitive**

- **Fracamente Tipada**

- ao declarar as variáveis não definimos o tipo (inteira, ponto flutuante, Cadeia, booleana, etc).

# GO

- Linguagem do GOOGLE
- Criada em 2007 por Rob Pike, Ken Thompson e Robert Griesemer
- Lançada mundialmente em 2009



# GO



**Desempenho**

**Escalabilidade**

**Facilidade de  
manutenção**

# GO - CARACTERÍSTICAS

- Linguagem específica
- Desempenho
- Compilação rápida
- Multiplataforma
- Facil compartilhamento
- Open Source
- Aplica os pilares da programação orientada a objetos
- Economiza os recursos de hardware

# HASKELL

- Haskell foi lançada em 1990, e recebeu este nome em homenagem ao matemático Haskell Brooks Curry.
- Bastante utilizada para pesquisas no meio acadêmico. É uma linguagem que possui foco no alcance de soluções para problemas matemáticos.




# HASKELL

- PARADIGMA DE PROGRAMAÇÃO
  - Puramente funcional
    - Apesar de existirem hoje implementações de conceitos OOP
    - Haskell++
  - Programar nesta linguagem é como encadear funções

# HASKELL - CARACTERÍSTICAS

- Estilo declarativo:
  - É usado um casamento de padrões através de expressões regulares
- Compilada:
  - O código fonte é transformado em código binário que será executado diretamente pelo hardware da plataforma.
- Fortemente Tipada:
  - Quando se declara uma variável em Haskell ela vai continuar sendo daquele tipo até o fim do programa.

A decorative wavy line in light blue and white on the left side of the slide.

# QUAIS OUTRAS LINGUAGENS DE PROGRAMAÇÃO VOCÊS CONHECEM?

QUAIS SUAS CARACTERÍSTICAS