
**Disciplina:** Linguagens de Programação

**Professor:** Adriano Benigno Moreira

benigno@uit.br


# Ementa

Conceitos de Linguagens de Programação. Paradigmas de Programação. Sintaxe e semântica. Sistemas de tipos. Tendências em Linguagens de Programação.


# Bibliografia

## Básica

1. MENEZES, Paulo Blauth. Linguagens formais e autômatos. 6. Porto Alegre Bookman 2011 ISBN 9788577807994.

2. SEBESTA, Robert W.. Conceitos de linguagens de programação. 11. ed. Porto Alegre, RS: Bookman, 2018. E-book (757 p.) ISBN 9788582604694.

## Complementar

1. WATT, David A.; FINDLAY, William. Programming language design concepts. Hoboken: J. Wiley, c2004. xviii, 473 p. ISBN 0470853204

Universidade de Itaúna

# Bibliografia

1. SEBESTA, Robert W.. Conceitos de linguagens de programação. 11. ed. Porto Alegre, RS: Bookman, 2018. E-book (757 p.) ISBN 9788582604694.

Universidade de Itaúna

# Avaliação

**1 - Trabalho 1 - 20 pontos**

**Trabalho 2 – 10 pontos**

**2 - Trabalho 1 - 20 pontos**

**Trabalho 2 – 10 pontos**

**3 - Prova 40 pontos**

**Total 40**

**Total Geral 100 pontos**

Universidade de Itaúna

# Linguagem de programação

## Introdução

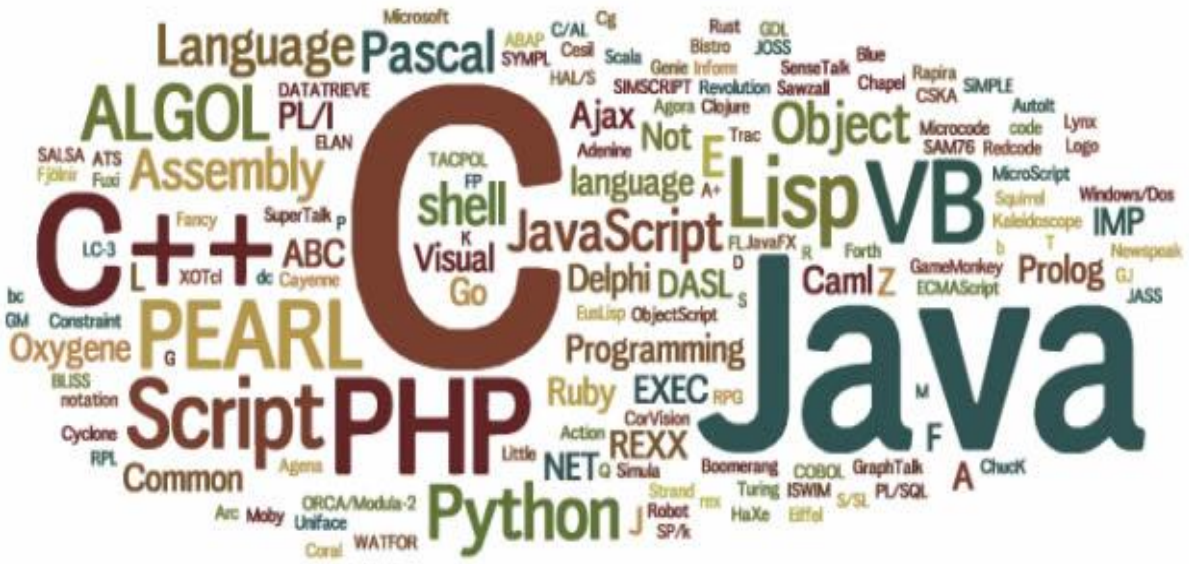
Uma linguagem de programação é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

O conjunto de palavras, compostas de acordo com essas regras da linguagem, constituem o código fonte de um software. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador.

Vídeo

Lingugens mais discutidas no momento em projetos

Importante



Universidade de Itaúna

# 1. Linguagem de programação

## Curiosidades

Exemplo: Programa "fatorial" escrito para o Pentium

```
.file   "fact.c"
        .section    .rodata
.LC0:   .string "> "
.LC1:   .string "%d"
.LC2:   .string "fact(%d) = %d\n"
.text
.globl main
.type main, @function
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        andl    $-16, %esp
        movl    $0, %eax
        addl    $15, %eax
        addl    $15, %eax
        shrl    $4, %eax
        sall    $4, %eax
        subl    %eax, %esp
        movl    $.LC0, (%esp)
        call    printf
        leal    -12(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC1, (%esp)
        call    scanf
        movl    $1, -4(%ebp)
        movl    $1, -8(%ebp)
        jmp     .L2
.L3:    movl    -4(%ebp), %eax
        imull   -8(%ebp), %eax
        movl    %eax, -4(%ebp)
        leal    -8(%ebp), %eax
        addl    $1, (%eax)
.L2:    movl    -12(%ebp), %eax
        cmpl    %eax, -8(%ebp)
        jle     .L3
        movl    -12(%ebp), %edx
        movl    -4(%ebp), %eax
        movl    %eax, 8(%esp)
        movl    %edx, 4(%esp)
        movl    $.LC2, (%esp)
        call    printf
        movl    $0, %eax
        leave
        ret

.size main, .-main
```

Exemplo: Programa "fatorial" escrito em Fortran

```
    READ 10,I
10  FORMAT(I3)
    J=1
    DO 20 K=1,I
    J=J*K
20  CONTINUE
    PRINT 10,J


    STOP
```

# Aplicação de uma linguagem

- Linguagens devem satisfazer restrições impostas pelas configurações técnicas nas quais elas são usadas
- Níveis de abstração na composição

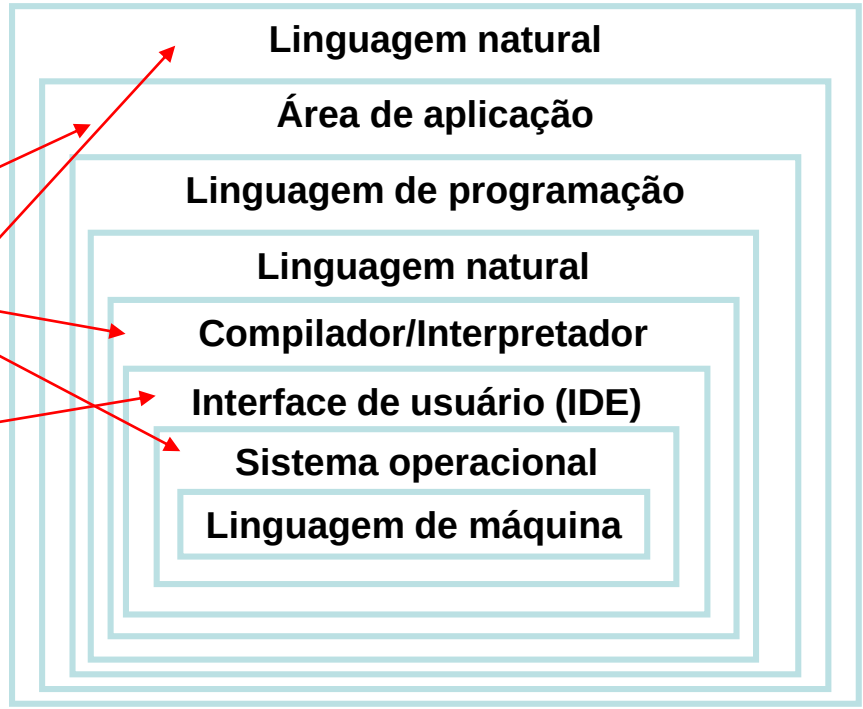
Fortran é usado em diferentes plataformas

Em diferentes compiladores

Para se adaptar as necessidades de programas científicos esse programadores podem trabalhar em diversas profissões

Que usam seus próprios projetos e ferramentas

E sua linguagem natural para comunicar-se entre eles



Universidade de Itaúna

# Aplicação de uma linguagem

Sempre é importante verificar as restrições de uma linguagem como
- Área de aplicação;
- Sistema operacional;
- IDE (Integrated Development Enviroment);
- Outras preferências de uma determinada comunidade.

| Linguagem | Propósito |
|---|---|
| **Ada** | Ser útil para todas as aplicações suportadas pelo departamento de defesa dos Estados Unidos |
| **Cobol** | Suportar todas as aplicações orientadas a negócios |
| **Prolog** | Processamento de linguagem natural, provas de teoremas e sistemas especializados. |
| **C** | Programação de sistemas operacionais |

# Aplicação de uma linguagem

## Padronização

Vídeo

O objetivo é estabelecer uma ordem, um padrão e grupos de programa, visando:

- Portabilidade;
- Estabilidade do sistema;
- Estrutura de linguagem.

O processo de padronização pode ser lento e sofre crítica por alguns.

Atualmente tempos entidades que se comprometem e estabelecer os padrões.

ANSI
American National Standards Institute

A American National Standards Institute, também conhecida por sua sigla ANSI, é uma organização particular Estado-Unidense sem fins lucrativos que tem por objetivo facilitar a padronização dos trabalhos de seus membros.

ISO
International Organization for Standardization

A ISO é uma organização não governamental que compreende órgãos de normalização de mais de 160 países, com um órgão de normalização representando cada país membro.

Universidade de Itaúna

# 1.1 Razões para estudar conceitos de linguagens de programação

- ***Aumento da capacidade de expressar ideias***. Os programadores podem aumentar a diversidade de seus processos mentais de desenvolvimento de software ao aprender novas construções de linguagens. Conhecer uma variedade mais ampla de recursos das linguagens de programação pode reduzir as limitações no desenvolvimento de software.
- ***Embasamento para escolher linguagens adequadas.*** Alguns programadores profissionais tiveram pouca educação formal em ciência da computação – em vez disso, desenvolveram suas habilidades em programação independentemente ou em programas de treinamento em suas empresas. Tais programas normalmente limitam o ensino a uma ou duas linguagens diretamente relevantes para os projetos atuais da organização. Outros programadores receberam seu treinamento formal há anos. As linguagens que aprenderam na época não são mais usadas e muitos recursos agora disponíveis não eram amplamente conhecidos então.

# 1.1 Razões para estudar conceitos de linguagens de programação

- ***Aumento da habilidade para aprender novas linguagens***. A programação de computadores ainda é uma disciplina relativamente nova e as metodologias de projeto, ferramentas de desenvolvimento de software e linguagens de programação ainda estão em evolução. Isso torna o desenvolvimento de software uma profissão empolgante, mas também exige aprendizado contínuo. O processo de aprender uma nova linguagem de programação pode ser longo e difícil, especialmente para alguém que se sinta à vontade com apenas uma ou duas e que nunca examinou os conceitos de linguagens de programação de um modo geral.

- ***Melhor entendimento da importância da implementação***. Ao aprender os conceitos de linguagens de programação, é tão interessante quanto necessário abordar aspectos de implementação que afetam esses conceitos. Em alguns casos, um entendimento de questões de implementação leva ao motivo pelo qual as linguagens foram projetadas de determinada forma. Por sua vez, esse conhecimento leva à habilidade de usar uma linguagem de maneira mais inteligente, conforme foi projetada para ser usada. Podemos ser programadores melhores ao entender as escolhas entre construções de linguagens de programação e as consequências dessas escolhas.

# 1.1 Razões para estudar conceitos de linguagens de programação

- ***Melhor uso de linguagens já conhecidas***. Em sua maioria, as linguagens de programação contemporâneas são extensas e complexas. É incomum um programador conhecer e usar todos os recursos da linguagem que utiliza. Ao estudar os conceitos de linguagens de programação, os programadores podem aprender sobre partes antes desconhecidas e não utilizadas das linguagens com que já trabalham e começar a utilizá-las.

- ***Avanço geral da computação.*** Por fim, existe uma visão geral de computação que pode justificar o estudo de conceitos de linguagens de programação. Apesar de normalmente ser possível determinar por que determinada linguagem se tornou popular, muitos acreditam, ao menos em retrospecto, que as linguagens de programação mais populares nem sempre são as melhores disponíveis. Em alguns casos, pode-se concluir que uma linguagem se tornou amplamente usada, ao menos em parte, porque aqueles em posições de escolha não estavam suficientemente familiarizados com conceitos de linguagens de programação.

# 1.2 Domínios de programação

Os computadores são utilizados em uma infinidade de tarefas, desde controlar usinas nucleares até disponibilizar jogos eletrônicos em telefones celulares. Por causa dessa diversidade de uso, linguagens de programação com objetivos muito diferentes foram desenvolvidas.

- ***1.2.1 Aplicações científicas***. Os primeiros computadores digitais, surgidos no final dos anos 1940, início dos anos 1950, foram desenvolvidos e usados para aplicações científicas. Normalmente, as aplicações científicas daquela época utilizavam estruturas de dados relativamente simples, mas exigiam diversos cálculos aritméticos de ponto flutuante. As estruturas de dados mais comuns eram os vetores e as matrizes; as estruturas de controle mais comuns eram os laços de contagem e as seleções. As primeiras linguagens de programação de alto nível inventadas para aplicações científicas foram projetadas para suprir tais necessidades.
- ***1.2.2 Aplicações empresariais.*** O uso de computadores para aplicações comerciais começou nos anos 1950. Computadores especiais foram desenvolvidos para esse propósito, com linguagens especiais. A primeira linguagem de alto nível para negócios a ser bem-sucedida foi o COBOL (ISO/IEC, 2002), com sua primeira versão aparecendo em 1960. COBOL provavelmente ainda é a linguagem mais utilizada para tais aplicações. Linguagens de negócios são caracterizadas por facilidades para a produção de relatórios elaborados, maneiras precisas de descrever e armazenar números decimais e caracteres, e a habilidade de especificar operações aritméticas decimais.

# 1.2 Domínios de programação

- ***1.2.3 Inteligência artificial*** A Inteligência Artificial (IA) é uma ampla área de aplicações computacionais caracterizadas pelo uso de computações simbólicas em vez de numéricas. Computações simbólicas são aquelas nas quais símbolos, compostos de nomes em vez de números, são manipulados. Além disso, a computação simbólica é feita de modo mais fácil por meio de listas ligadas de dados do que por meio de vetores. Esse tipo de programação algumas vezes requer mais flexibilidade do que outros domínios de programação. Por exemplo, em algumas aplicações de IA, a capacidade de criar e executar segmentos de código durante a execução é conveniente. A primeira linguagem de programação amplamente utilizada, desenvolvida para aplicações de IA, foi a linguagem funcional Lisp, que apareceu em 1959. A maioria das aplicações de IA desenvolvidas antes de 1990 foi escrita em Lisp ou em um de seus parentes próximos. No início dos anos 1970, entretanto, uma abordagem alternativa a algumas dessas aplicações apareceu – programação lógica usando a linguagem Prolog.

# 1.2 Domínios de programação

- ***1.2.3 A World Wide Web*** é mantida por uma eclética coleção de linguagens, que vão desde linguagens de marcação, como HTML, que não é de programação, até linguagens de programação de propósito geral, como Java. Dada a necessidade universal de conteúdo dinâmico na Web, alguma capacidade de computação geralmente é incluída na tecnologia de apresentação de conteúdo. Essa funcionalidade pode ser fornecida por código de programação embarcado em um documento HTML. Tal código é normalmente escrito com uma linguagem de scripting, como JavaScript ou PHP (Tatroe, 2013).

# 1.3 Critérios de avaliação de linguagens

Existem diversos critérios que podem ser adotados a fim de fazer uma avaliação de uma linguagem de programação, dificilmente os cientistas da computação entram em consenso sobre quais são os critérios mais adequados para fazer avaliações da sintaxe das linguagens, porém alguns itens básicos são praticamente um consenso entre quase todos. Estes são os apresentados a seguir, na Tabela 1 que mostra em quais critérios as características de uma linguagem se enquadra.

TABELA 1.1 Critérios de avaliação de linguagens e as características que os afetam

| | **Critérios** | | |
|---|---|---|---|
| **Característica** | Legibilidade | Facilidade de Escrita | Confiabilidade |
| Simplicidade | X | X | X |
| Ortogonalidade | X | X | X |
| Instruções de Controle | X | X | X |
| Tipos de dados | X | X | X |
| Projeto da Sintaxe | X | X | X |
| Suporta a abstração | | X | X |
| Expressividade | | X | X |
| Verificação de tipos | | | X |
| Tratamento de exceções | | | X |
| Ponteiros Restritos | | | X |

# 1.3 Critérios de avaliação de linguagens

- ***1.3.1 LEGIBILIDADE:***

Sem dúvida um dos critérios de avaliação de linguagens de programação mais importantes é o da legibilidade, que é o grau de facilidade com que os algoritmos criados naquela linguagem podem ser lidos e entendidos. A facilidade de manutenção de aplicações está diretamente ligada à legibilidade do código, por isto essa se torna um dos critérios mais importantes de avaliação. A legibilidade de um programa também está ligada ao domínio do problema em questão, pois quando determinada linguagem de programação não foi projetada para determinado tipo de problema, é bastante provável que seu código fique confuso, tornando difícil de ser lido e interpretado. Por exemplo, se um programa que descreve um cálculo é escrito em uma linguagem que não foi projetada para tal uso, ele pode não ser natural e ser desnecessariamente complexo, tornando complicada sua leitura (quando, em geral, seria algo simples).

**Qualquer tolo pode escrever código que um computador pode entender. Bons programadores escrevem código que seres humanos podem entender**. – *Martin Fowler*.

# 1.3 Critérios de avaliação de linguagens

- ***1.3.1.1 Simplicidade Geral:*** A simplicidade de uma linguagem de programação está diretamente ligada a sua legibilidade, isto porque uma linguagem com uma sintaxe extensa é muito mais difícil de ser aprendida do que uma linguagem que possua poucos artefatos básicos. A maioria dos desenvolvedores geralmente aprende apenas parte de uma linguagem de programação, nunca ela no seu todo, ignorando alguns recursos. Este tipo de atitude é tomada principalmente para que não seja necessário um estudo mais profundo da linguagem, tornando o início do desenvolvimento rápido, trazendo resultados também rápidos. O problema desta abordagem é que outros desenvolvedores podem ter aprendido outra parte da linguagem, diferente daquela que aprendeu o desenvolvedor do sistema, tornando difícil a compreensão do código posteriormente.

  Por exemplo, em Java um usuário pode incrementar uma simples variável inteira de quatro maneiras:

```
count = count + 1
count += 1
count++
++count
```

  Apesar de as últimas duas sentenças terem significados um pouco diferentes uma da outra e em relação às duas primeiras em alguns contextos, todas elas têm o mesmo significado quando usadas em expressões isoladas.

# 1.3 Critérios de avaliação de linguagens

- ***1.3.1.2 Ortogonalidade:*** conjunto consistente de regras para combinar construções primitivas, com poucas exceções. Torna as linguagens fáceis de leitura e aprendizagem. O significado de um recurso da linguagem é independente do contexto. Um conjunto relativamente pequeno de construções primitivas podem ser combinadas em um número relativamente pequeno de maneiras para construir as estruturas de dados e controle da linguagem. Qualquer combinação possível das construções primitivas é aceita.
  - Instruções de controle.
  - Tipos e estruturas de dados;
  - Sintaxe: tamanho dos identificadores. Uso de palavras reservadas.

***1.3.1.3 Tipo de Dados***: A presença de mecanismos adequados para definir tipos e estruturas de dados é outro auxílio significativo à legibilidade. Por exemplo, suponha que um tipo numérico seja usado como uma flag porque não existe nenhum tipo booleano na linguagem. Em tal linguagem, poderíamos ter uma atribuição como:

```
timeOut = 1
```

O significado dessa sentença não é claro. Em uma linguagem que inclui tipos booleanos, teríamos:

```
timeOut = true
```

O significado dessa sentença é perfeitamente claro.

# 1.3 Critérios de avaliação de linguagens

- ***1.3.1.4 Projeto da sintaxe:*** A sintaxe, ou forma, dos elementos de uma linguagem tem um efeito significativo na legibilidade dos programas. A seguir, estão alguns exemplos de escolhas de projeto sintáticas que afetam a legibilidade.
  - **Palavras especiais.** A aparência de um programa e sua legibilidade são muito influenciadas pela forma das palavras especiais de uma linguagem (por exemplo, while, class e for).

- ***1.3.2 FACILIDADE DE ESCRITA:***

  A facilidade de escrita é a medida do quão facilmente uma linguagem pode ser usada para criar programas para um domínio. A maioria das características de linguagem que afetam a legibilidade também afetam a facilidade de escrita. Como ocorre com a legibilidade, a facilidade de escrita deve ser considerada no contexto do domínio de problema alvo de uma linguagem. Não é justo comparar a facilidade de escrita de duas linguagens no contexto de determinada aplicação quando uma delas foi projetada para tal aplicação e a outra não.

# 1.3 Critérios de avaliação de linguagens

- ***1.3.2.1 A expressividade:*** em uma linguagem pode se referir a diversas características. Em uma linguagem expressividade significa a existência de operadores muito poderosos que permitem muitas computações com um programa muito pequeno. Em geral, uma linguagem expressiva especifica computações de uma forma conveniente, em vez de deselegante. Por exemplo, em C, a notação **count++** é mais conveniente e menor que **count = count + 1**. A inclusão da sentença **for** em Java torna a escrita de laços de contagem mais fácil do que com o uso do **while**, também possível. Todas essas construções aumentam a facilidade de escrita de uma linguagem.
  - Expressividade x concisão
    - muito concisa: falta expressividade?
    - muito extensa: falta simplicidade?

# 1.3 Critérios de avaliação de linguagens

- ***1.3.3*** **CONFIABILIDADE**

  Diz-se que um programa é confiável quando está de acordo com suas especificações em todas as condições.

  ***1.3.3.1*** **Verificação de tipos*:*** é a execução de testes para detectar erros de tipos em um programa, tanto por parte do compilador quanto durante a execução de um programa. Ela é um fator importante na confiabilidade de uma linguagem. Como a verificação de tipos em tempo de execução é dispendiosa, a verificação em tempo de compilação é mais desejável. Além disso, quanto mais cedo os erros nos programas forem detectados, mais barato será fazer todos os reparos necessários. O projeto de Java requer verificações dos tipos de praticamente todas as variáveis e expressões em tempo de compilação. Isso praticamente elimina erros de tipos em tempo de execução em programas Java.

  **1.3.3.2 Tratamento de exceções:** A habilidade de um programa de interceptar erros em tempo de execução (além de outras condições não usuais detectáveis pelo programa), tomar medidas corretivas e então continuar melhora a confiabilidade. Tal facilidade é chamada de tratamento de exceções. Ada, C++, Java e C# contêm recursos extensivos para tratamento de exceções, mas tais funções são praticamente inexistentes em algumas linguagens amplamente utilizadas, por exemplo C.

# 1.3 Critérios de avaliação de linguagens

**1.3.3.3 Apelidos:** Em uma definição bastante informal, apelidos são utilizados quando é possível ter um ou mais nomes em um programa para acessar a mesma célula de memória. Atualmente, é geralmente aceito que o uso de apelidos é um recurso perigoso em uma linguagem de programação. A maioria das linguagens permite algum tipo de apelido – por exemplo, dois ponteiros (ou referências) configurados para apontar para a mesma variável, o que é possível na maioria das linguagens. O programador deve sempre lembrar que trocar o valor apontado por um dos dois ponteiros modifica o valor referenciado pelo outro.

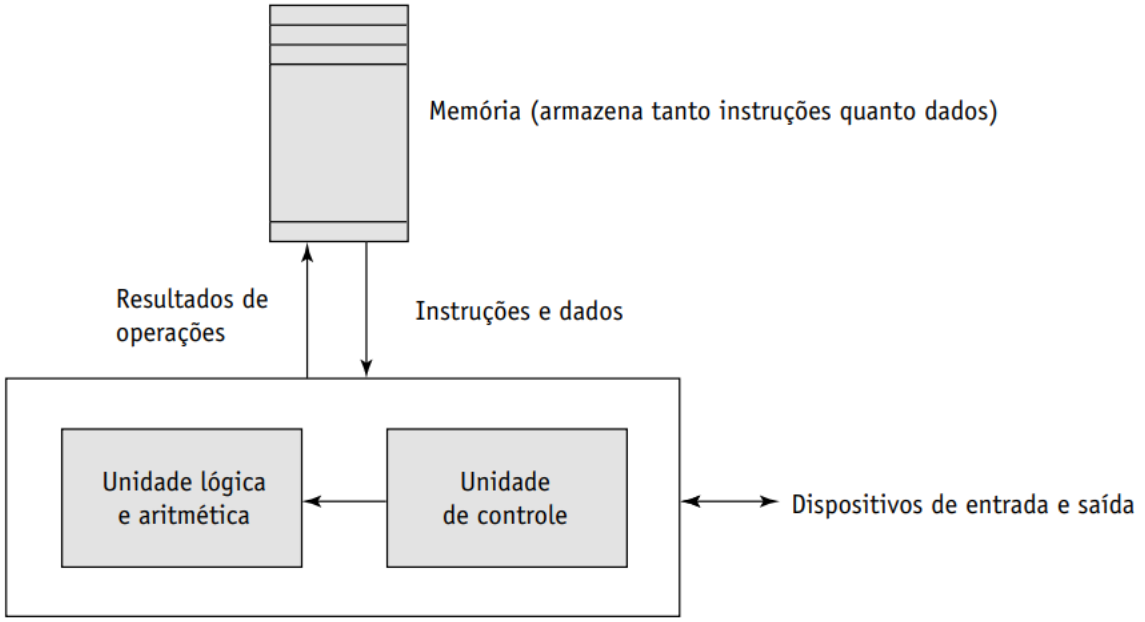
**1.3.3.4 Legibilidade e facilidade de escrita:** A legibilidade afeta a confiabilidade tanto nas fases de escrita quanto nas de manutenção do ciclo de vida. Programas difíceis de ler são também difíceis de escrever e modificar. Quanto mais fácil é escrever um programa, maior a probabilidade de ele estar correto.

# 1.4 Influência no projeto de linguagens

- ***1.4.1 Arquitetura de computadores***

A arquitetura básica dos computadores tem um efeito profundo no projeto de linguagens. A maioria das linguagens populares dos últimos 60 anos tem sido projetada considerando a principal arquitetura de computadores, chamada de arquitetura de von Neumann, cujo nome homenageia um de seus criadores, John von Neumann Elas são chamadas de linguagens imperativas. Em um computador von Neumann, tanto os dados quanto os programas são armazenados na mesma memória. A unidade central de processamento (CPU), que executa instruções, é separada da memória. Logo, instruções e dados devem ser transmitidos da memória para a CPU. Resultados de operações na CPU devem ser retornados para a memória. Praticamente todos os computadores digitais construídos desde os anos 1940 são baseados nessa arquitetura. A execução de um programa em código de máquina em uma arquitetura de computadores von Neumann ocorre em um processo chamado de **ciclo de obtenção e execução.** Conforme mencionado programas residem na memória, mas são executados na CPU.

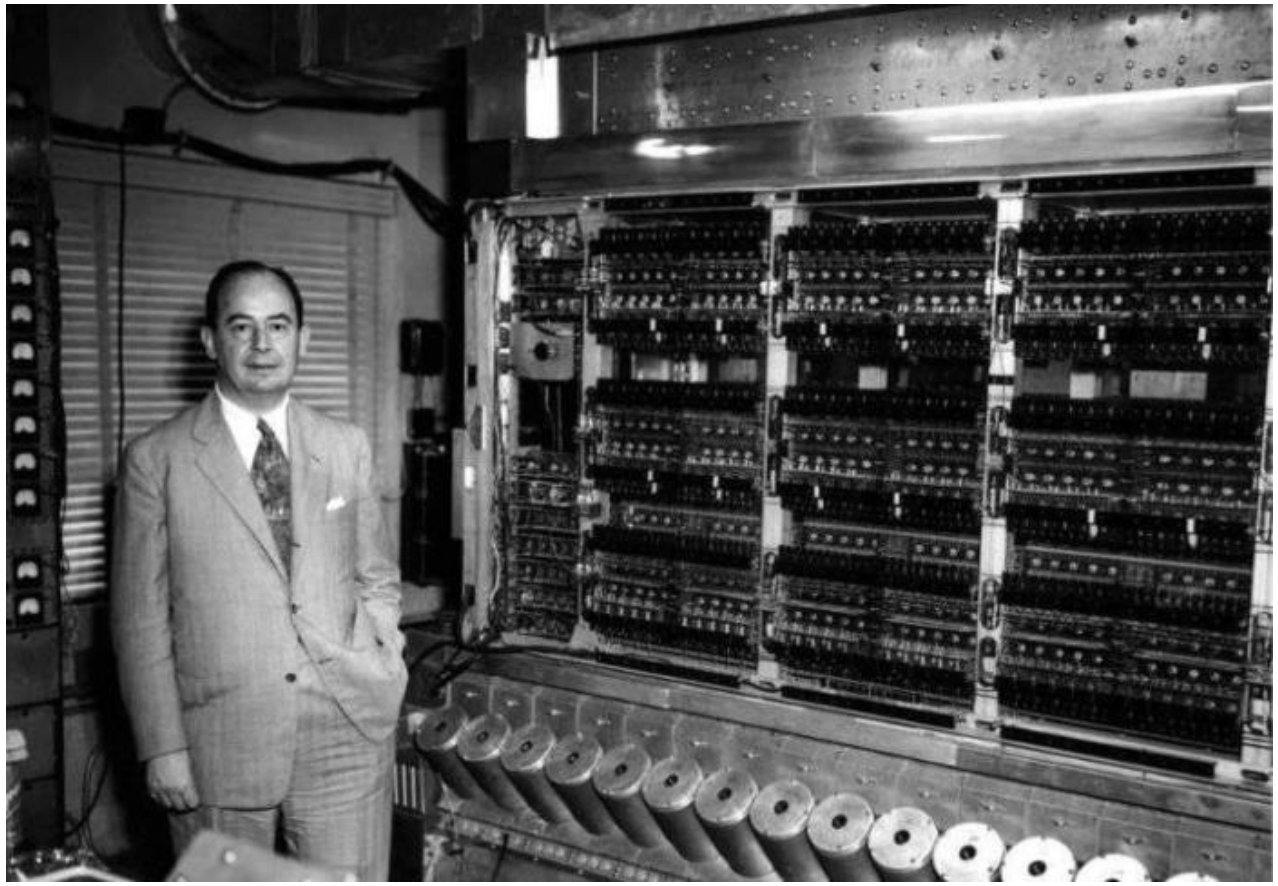
Vídeo




Unidade de processamento central

Universidade de Itaúna

# 1.4 Influência no projeto de linguagens



**John von Neumann**,
Nascido Margittai Neumann János Lajos (Budapeste, 28 de dezembro de 1903 — Washington, D.C., 8 de fevereiro de 1957) foi um matemático húngaro de origem judaica, naturalizado estadunidense.

# 1.4 Influência no projeto de linguagens

- ***1.4.2 Metodologia de projeto de programas***
  - **Anos 60**. O final dos anos 1960 e o início dos anos 1970 trouxeram uma análise intensa, iniciada em grande parte pelo movimento da programação estruturada, tanto do processo de desenvolvimento de software quanto do projeto de linguagens de programação. Em vez de simplesmente resolver conjuntos de equações para simular rotas de satélites, como no início dos anos 1960, os programas eram escritos para executar tarefas extensas e complexas, como controlar grandes refinarias de petróleo e fornecer sistemas de reservas de passagens aéreas em âmbito mundial.
  - **Anos 70**. As novas metodologias de desenvolvimento de software que emergiram como um resultado da pesquisa nos anos 1970 foram chamadas de projeto descendente (*top-down*) e de refinamento passo a passo. As principais deficiências descobertas nas linguagens de programação eram na verificação de tipos e a inadequação das sentenças de controle (que exigiam uso intenso de desvios incondicionais, também conhecidos como gotos).
  - **Anos 80**. O último grande passo na evolução do desenvolvimento de software começou no início dos anos 1980, é o projeto orientado a objetos. As metodologias orientadas a objetos começam com a abstração de dados, que encapsula o processamento com os objetos de dados, controla o acesso aos dados e também adiciona mecanismos de herança e vinculação dinâmica de métodos.

# 1.4 Categorias de Linguagens

- As linguagens de programação normalmente são divididas em quatro categorias: **imperativas, funcionais, lógicas e orientadas a objetos**. Entretanto, é difícil considera que linguagens que suportam a orientação a objetos formem uma categoria separada. As linguagens mais populares que suportam a orientação a objetos cresceram a partir de linguagens imperativas. Apesar de o paradigma de desenvolvimento de software orientado a objetos diferir significativamente do paradigma orientado a procedimentos usado normalmente nas linguagens imperativas, as extensões a uma linguagem imperativa, necessárias para dar suporte à programação orientada a objetos, não são intensivas. Por exemplo, as expressões, sentenças de atribuição e sentenças de controle de C e Java são praticamente idênticas.

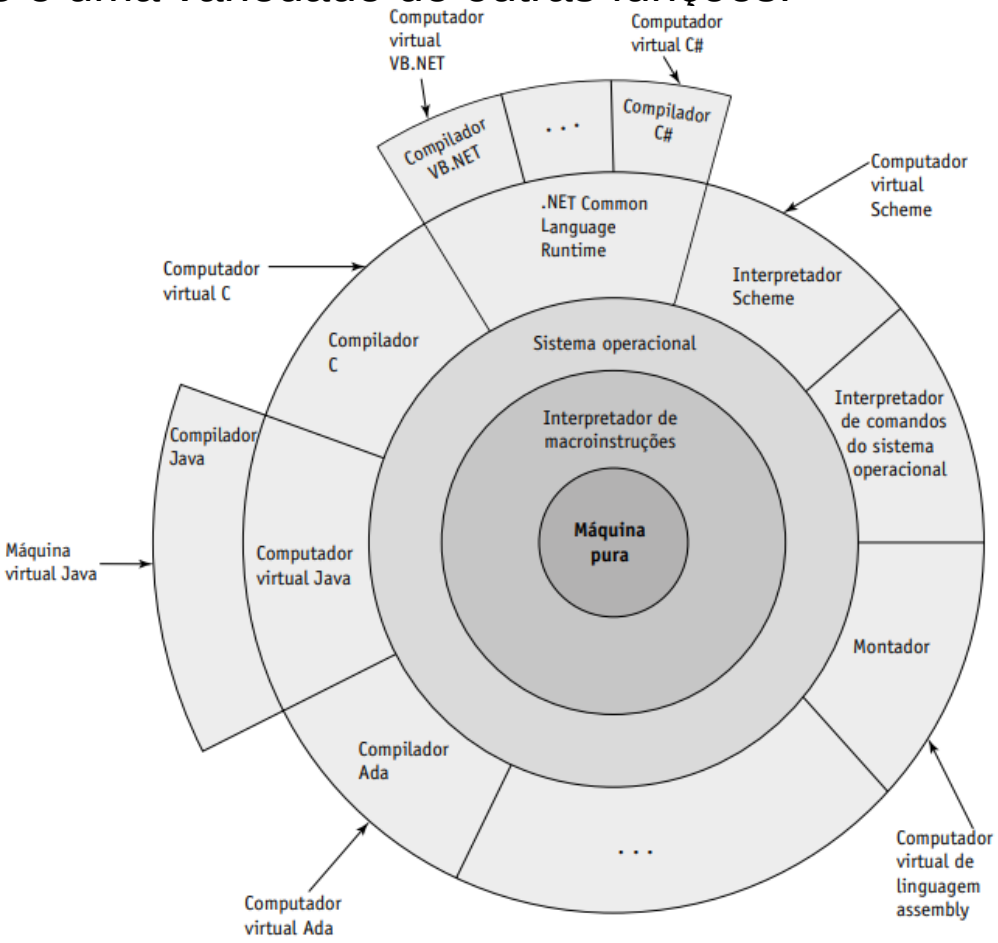
# 1.6 *Trade-offs* no projeto de linguagens

Em seu brilhante artigo sobre o projeto de linguagens, *Hoare (1973)* afirma que “existem tantos critérios importantes, mas conflitantes, que sua harmonização e satisfação estão entre as principais tarefas de engenharia”. Dois critérios conflitantes são a confiabilidade e o custo de execução. Por exemplo, a linguagem Java exige que todas as referências aos elementos de um vetor sejam verificadas para garantir que os índices estejam em suas faixas válidas. Esse passo aumenta muito o custo de execução de programas Java que contenham um grande número de referências a elementos de vetores. C não exige a verificação da faixa de índices – dessa forma, os programas em C executam mais rápido que programas semanticamente equivalentes em Java, apesar de estes serem mais confiáveis. Os projetistas de Java trocaram eficiência de execução por confiabilidade. Outro exemplo, o conflito entre a facilidade de escrita e a legibilidade é comum no projeto de linguagens. Os ponteiros de C++ podem ser manipulados de diversas maneiras, o que possibilita um endereçamento de dados altamente flexível. Devido aos potenciais problemas de confiabilidade com o uso de ponteiros, eles não foram incluídos em Java.

# 1.7 Métodos de implementação

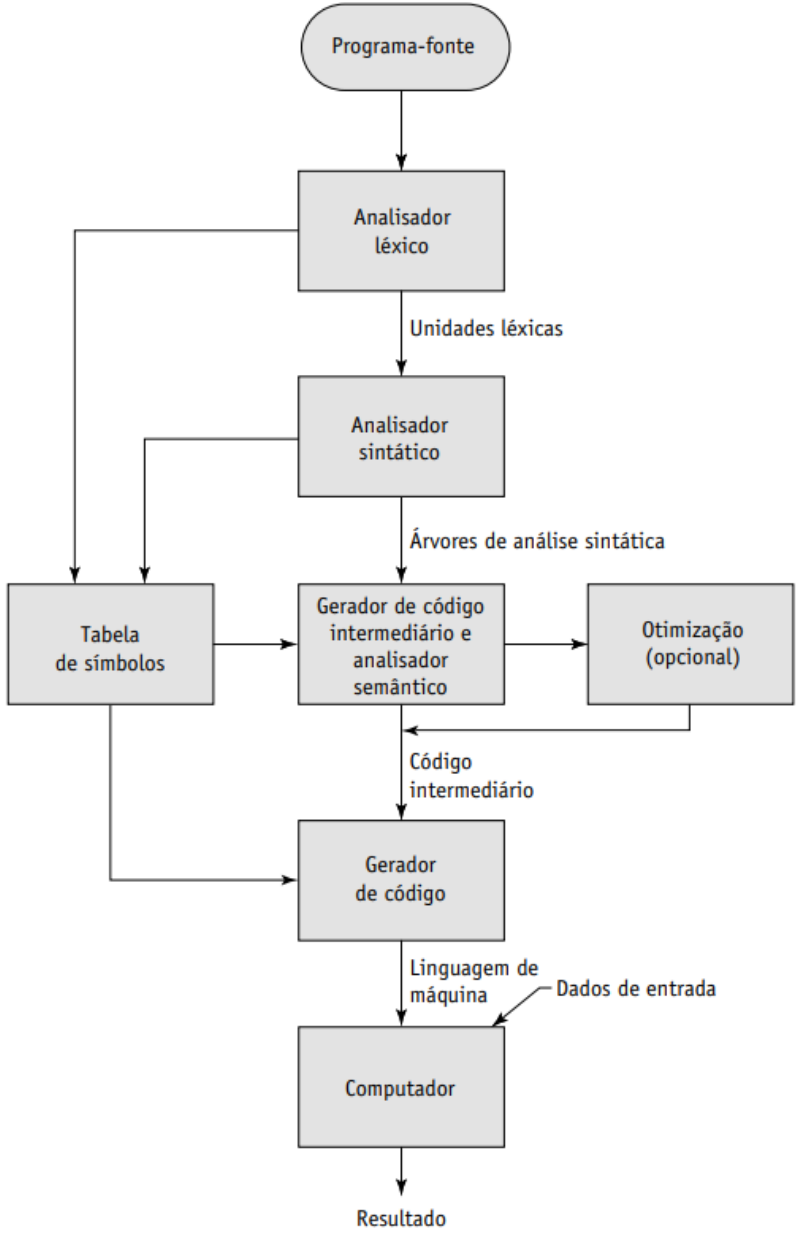
Um sistema de implementação de linguagem não pode ser o único aplicativo de software em um computador. Também é necessária uma grande coleção de programas, chamada de sistema operacional, a qual fornece primitivas de nível mais alto do que aquelas fornecidas pela linguagem de máquina. Essas primitivas fornecem funções para o gerenciamento de recursos do sistema, operações de entrada e saída, um sistema de gerenciamento de arquivos, editores de texto e/ou de programas e uma variedade de outras funções.

Como os sistemas de implementação de linguagens precisam de muitas das facilidades do sistema operacional, eles fazem uma interface com o sistema, em vez de diretamente com o processador (em linguagem de máquina). O sistema operacional e as implementações de linguagem são colocados em camadas superiores à interface de linguagem de máquina de um computador. Essas camadas podem ser vistas como computadores virtuais, fornecendo interfaces para o usuário em níveis mais altos. Por exemplo, um sistema operacional e um compilador C fornecem um computador C virtual. Com outros compiladores, uma máquina pode se tornar outros tipos de computadores virtuais. A maioria dos sistemas de computação fornece diferentes computadores virtuais. Os programas de usuários formam outra camada sobre a de computadores virtuais'.



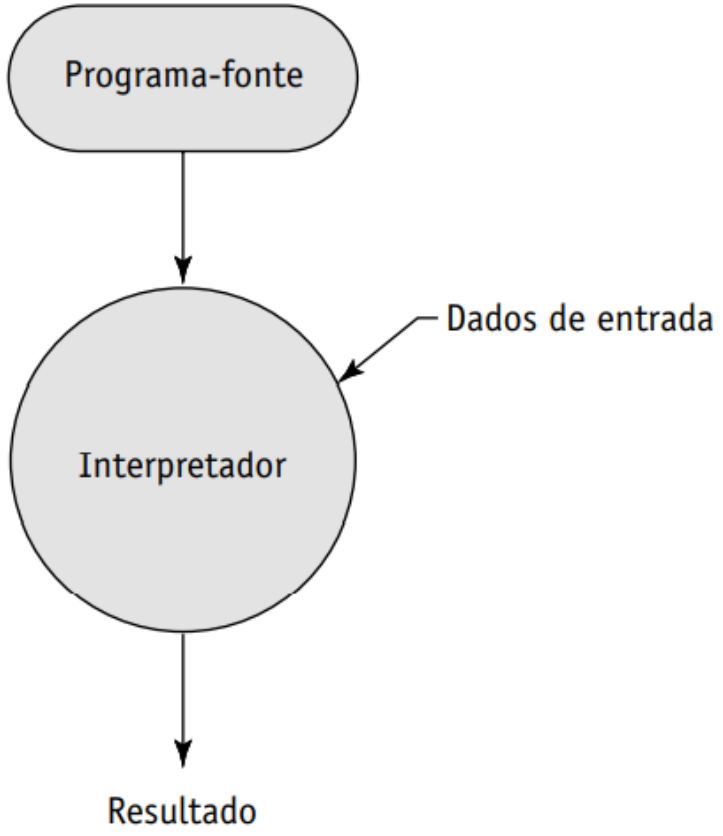
# 1.7 Métodos de implementação

- ***1.7.1 Compilação***
  - São programas que transformam códigos fontes em programas executáveis. A maioria das implementações de produção das linguagens, como C, COBOL e C++, é feita por meio de compiladores.
  - **O analisador léxico** reúne os caracteres do programa-fonte em unidades léxicas. As unidades léxicas de um programa são identificadores, palavras especiais, operadores e sinais de pontuação.
  - **O analisador sintático** obtém as unidades léxicas do analisador léxico e as utiliza para construir estruturas hierárquicas chamadas de **árvores de análise sintática** (*parse trees*). Essas árvores de análise sintática representam a estrutura sintática do programa. Em muitos casos, nenhuma estrutura de árvore de análise sintática é realmente construída; em vez disso, a informação que seria necessária para construir a árvore é gerada e usada diretamente.
  - **O gerador de código** intermediário produz um programa em uma linguagem diferente, em um nível intermediário entre o programa-fonte e a saída final do compilador: o programa em linguagem de máquina



# 1.7 Métodos de implementação

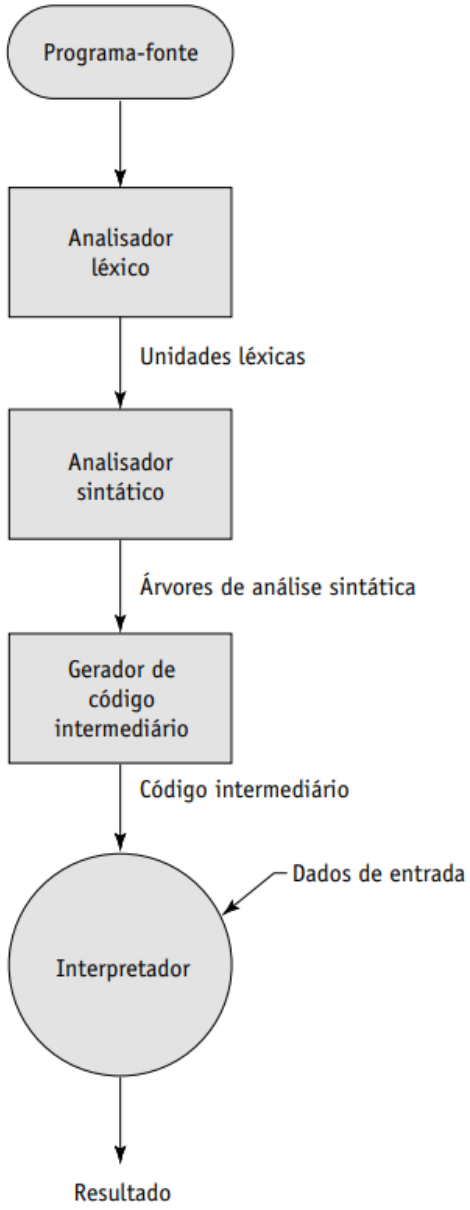
- ***1.7.2 Interpretação pura***
  - A interpretação pura reside na extremidade oposta (em relação à compilação) dos métodos de implementação. Com essa abordagem, os programas são interpretados por outro, chamado interpretador, sem tradução. A interpretação pura tem a vantagem de permitir uma fácil implementação de muitas operações de depuração em código-fonte, pois todas as mensagens de erro em tempo de execução podem referenciar unidades de código-fonte. Em contrapartida, esse método tem uma séria desvantagem em relação ao tempo de execução, que é de 10 a 100 vezes mais lento do que nos sistemas compilados. Além disso, independentemente de quantas vezes uma sentença for executada, ela deve ser decodificada a cada vez. Logo, a decodificação de sentenças, e não a conexão entre o processador e a memória, é o gargalo de um interpretador puro. Outra desvantagem da interpretação pura é que ela normalmente exige mais espaço.



# 1.7 Métodos de implementação

- ***1.7.3*** **Sistemas de implementação híbridos**
- Alguns sistemas de implementação de linguagens são um meio-termo entre os compiladores e os interpretadores puros; eles traduzem os programas em linguagem de alto nível para uma linguagem intermediária projetada para facilitar a interpretação. Esse método é mais rápido que a interpretação pura, porque as sentenças da linguagem-fonte são decodificadas apenas uma vez. Tais implementações são chamadas de **sistemas de implementação híbridos**.

  Perl é implementado como um sistema híbrido. Os programas em Perl são parcialmente compilados para detectar erros antes da interpretação e para simplificar o interpretador



# 1.7 Métodos de implementação

- ***1.7.4*** **Pré-processadores**
- Um pré-processador é um programa que processa outro programa imediatamente antes de ele ser compilado. As instruções de pré-processador são embutidas em programas. O pré-processador é essencialmente um programa que expande macros. As instruções de pré-processador são comumente usadas para especificar que o código de outro arquivo deve ser incluído. Por exemplo, a instrução de pré-processador de C o faz copiar o conteúdo de myLib.h no programa, na posição da instrução #include

```
#include "myLib.h"
```

# 1.8 Métodos de implementação

***A*mbientes de programação**

- Um ambiente de programação é a coleção de ferramentas usadas no desenvolvimento de software. Essa coleção pode consistir em apenas um sistema de arquivos, um editor de textos e um compilador. Ou pode incluir uma grande coleção de ferramentas integradas, cada uma acessada por meio de uma interface de usuário uniforme. Logo, as características de uma linguagem de programação não são a única medida da capacidade de desenvolvimento de um sistema.
- O UNIX é um ambiente de programação mais antigo, inicialmente distribuído em meados dos anos 1970, construído em torno de um sistema operacional de multiprogramação portável. Ele fornece uma ampla gama de ferramentas de suporte poderosas para a produção e a manutenção de software em uma variedade de linguagens. No passado, o recurso mais importante que não existia no UNIX era uma interface uniforme entre suas ferramentas.
- O JBuilder é um ambiente de programação que fornece compilador, editor, depurador e sistema de arquivos integrados para desenvolvimento em Java, onde todos são acessados por meio de uma interface gráfica. O JBuilder é um sistema complexo e poderoso para criar software em Java.
- O Microsoft Visual Studio .NET é um passo relativamente recente na evolução dos ambientes de desenvolvimento de software. Ele é uma grande e elaborada coleção de ferramentas de desenvolvimento, todas usadas por meio de uma interface baseada em janelas.
- O NetBeans é um ambiente usado primariamente para o desenvolvimento de aplicações com Java, mas também oferece suporte para JavaScript, Ruby e PHP.

# Referência


- SEBESTA, Robert W.. Conceitos de linguagens de programação. 11. ed. Porto Alegre, RS: Bookman, 2018. E-book (757 p.) ISBN 9788582604694.