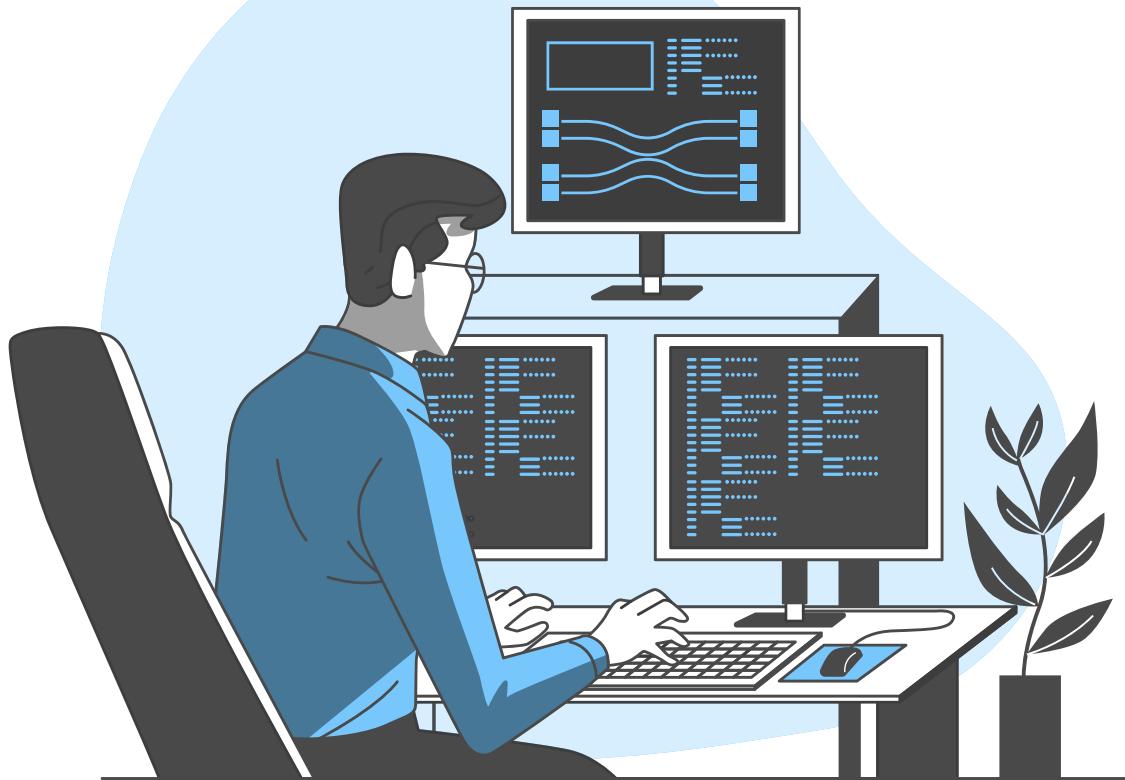


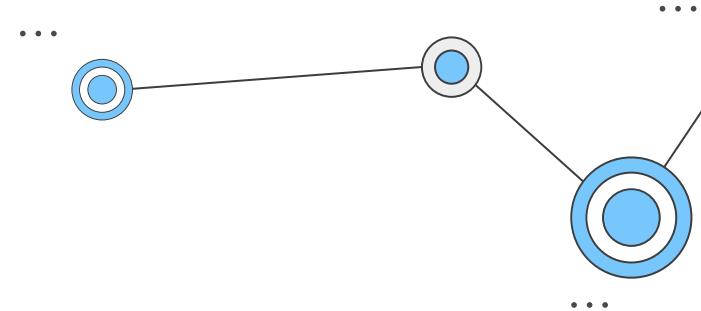


PUC Minas



Algoritmos e Estruturas de Dados I

Prof. Dr. João Paulo Aramuni



Aula 01

Nivelamento

AEDS I - Manhã

Nivelamento

- 1.1 Introdução a algoritmos e estruturas de dados
- 1.2 Aplicações de algoritmos
- 1.3 Modelos, notações e representações
- 1.4 Noções de linguagens de programação
- 1.5 Conceitos e utilizações
- 1.6 Representação de dados
- 1.7 Boas práticas: noções de documentação, contagem de operações e metodologia para desenvolvimento e testes

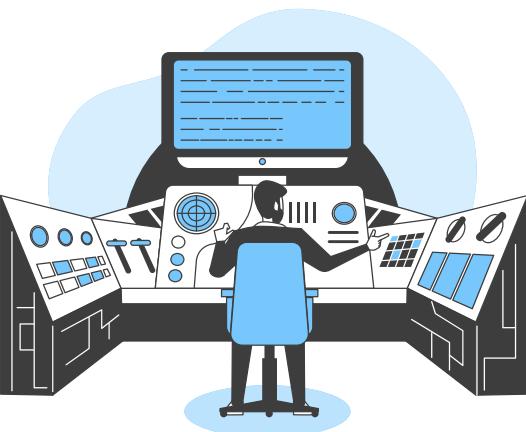
Unidade de Ensino



1.1 Introdução a algoritmos e estruturas de dados

Sempre que queremos resolver um problema no computador precisamos, antes de tudo, descrevê-lo de forma clara e precisa. Ou seja, precisamos escrever o seu **algoritmo**.

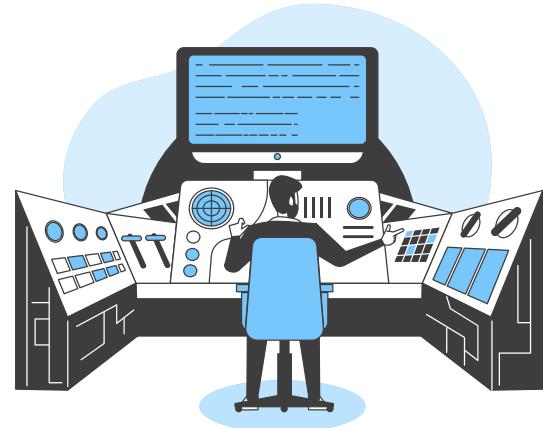
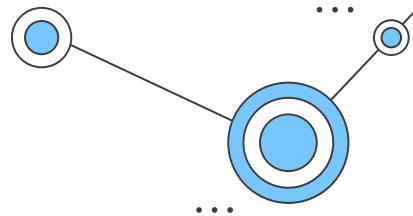
Algoritmo pode ser definido como uma sequência simples e objetiva de instruções para solucionar um determinado problema. Cada instrução é uma informação que indica ao computador uma ação básica a ser executada.



1.1 Introdução a algoritmos e estruturas de dados

Um **algoritmo** é uma sequência de instruções que realizam uma tarefa específica e é frequentemente ilustrado como uma receita, por exemplo, de bolo:

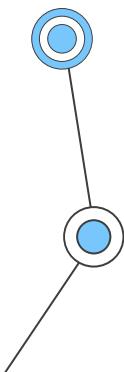
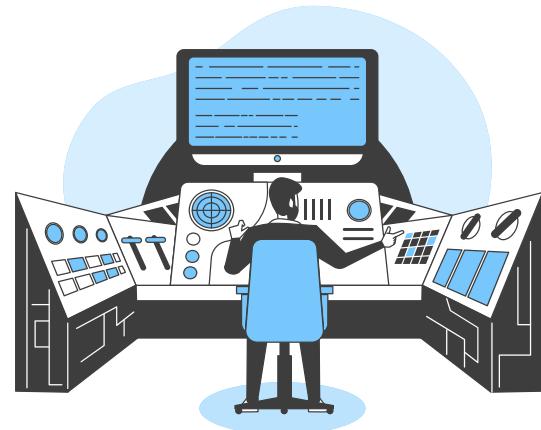
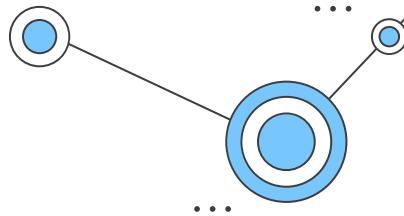
- Aqueça o forno à 180°C;
- unte uma forma redonda;
- numa taça bata até ficar cremoso:
 - 75g de manteiga;
 - 250g de açúcar;
- junte:
 - 4 ovos, um a um;
 - 100g de chocolate derretido;
- adicione aos poucos 250g de farinha peneirada;
- deite a massa na forma;
- leve ao forno durante 40 minutos.



1.1 Introdução a algoritmos e estruturas de dados

A sequência de instruções que define o algoritmo deve ser sempre **finita** e também não pode ser ambígua.

Isso significa que nosso algoritmo deve ter sempre um fim (ou seja, deve terminar após determinado número de passos) e que cada instrução deve ser precisamente definida (ou seja, não deve permitir mais de uma interpretação de seu significado).

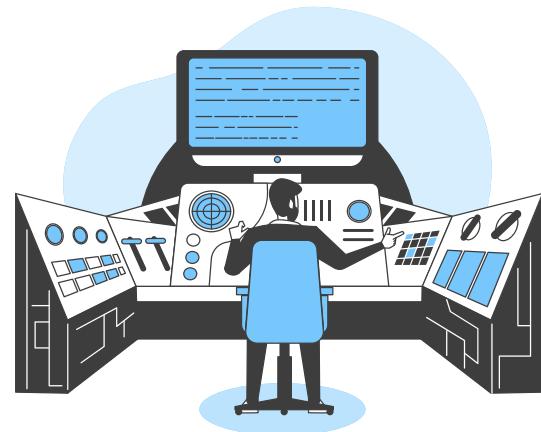


1.1 Introdução a algoritmos e estruturas de dados

Além disso, os algoritmos se baseiam no uso de um conjunto de instruções bem definido, que constituem um **vocabulário** de símbolos limitado (por exemplo, o conjunto de palavras reservadas da linguagem).

Exemplos na Linguagem C:

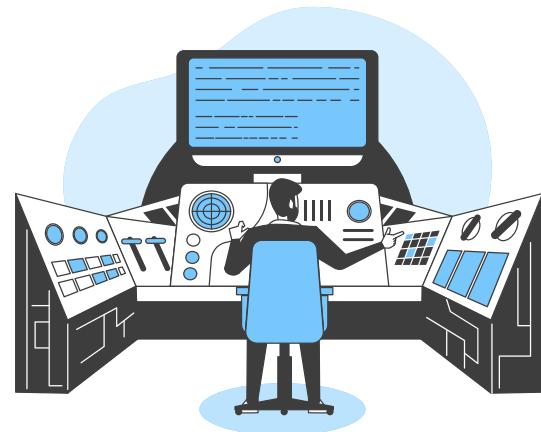
- char, float, long, int, if, else, for, while, void...



1.1 Introdução a algoritmos e estruturas de dados

Um algoritmo é um procedimento computacional composto de 3 partes:

- **Entrada de dados:** são os dados do algoritmo informados pelo usuário;
- **Processamento de dados:** são os procedimentos utilizados para chegar ao resultado. É responsável pela obtenção dos dados de saída com base nos dados de entrada;
- **Saída de dados:** são os dados já processados, apresentados ao usuário.

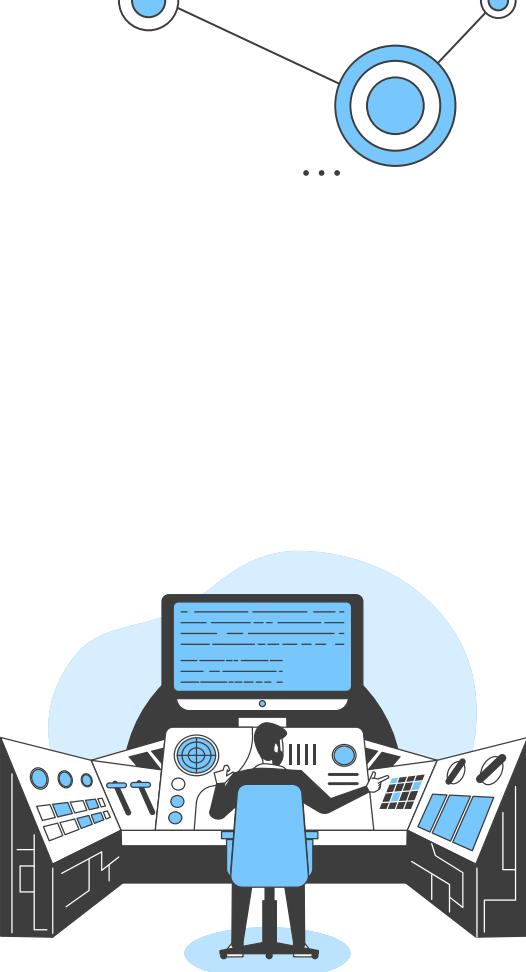


1.1 Introdução a algoritmos e estruturas de dados

Exemplo de um algoritmo simples em C para somar dois números inteiros:

```
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 int soma(int a, int b){
15     int c = a + b;
16     return c;
17 }
18
19 int main(void) {
20     puts("!!!Hello World!!!");
21     int x = 2;
22     int y = 3;
23     int z = soma(x,y);
24     printf("Soma = %d\n", z);
25
26     return EXIT_SUCCESS;
27 }
```

```
Console X Problems Tasks Properties
<terminated> (exit value: 0) Prj_HelloWorld [C/C++ Application] /Users/joaopauloaramuni/Documents/
!!!Hello World!!!
Soma = 5
```

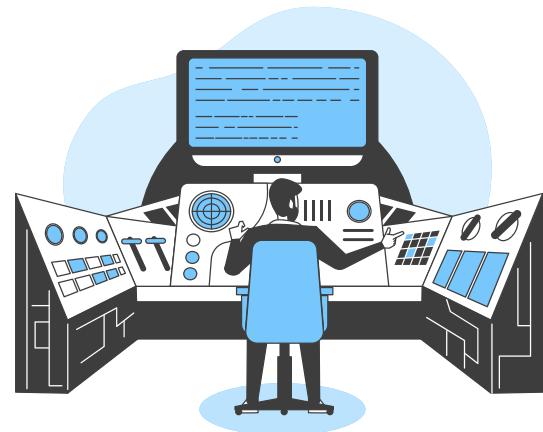
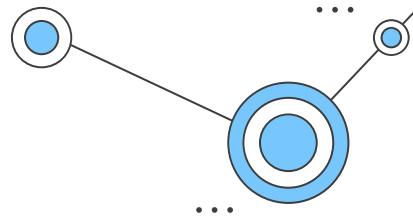


1.1 Introdução a algoritmos e estruturas de dados

Muitas vezes, um mesmo problema pode ser resolvido por vários algoritmos diferentes.

Os algoritmos se diferenciam um dos outros pela maneira como eles utilizam os recursos do computador. Existem algoritmos que dependem principalmente do tempo que demoram para serem executados, enquanto outros dependem da quantidade de memória do computador.

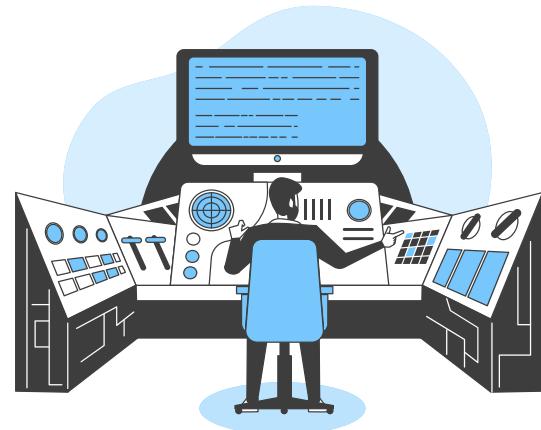
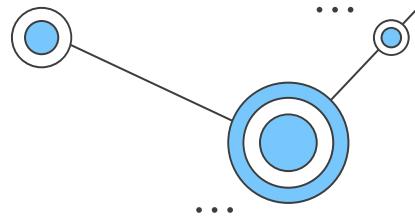
Um exemplo de problema em que existem vários algoritmos é a **ordenação de números**.



1.1 Introdução a algoritmos e estruturas de dados

Já uma **estrutura de dados**, em computação, é uma forma de armazenar e organizar os dados de modo que eles possam ser usados de forma **eficiente**.

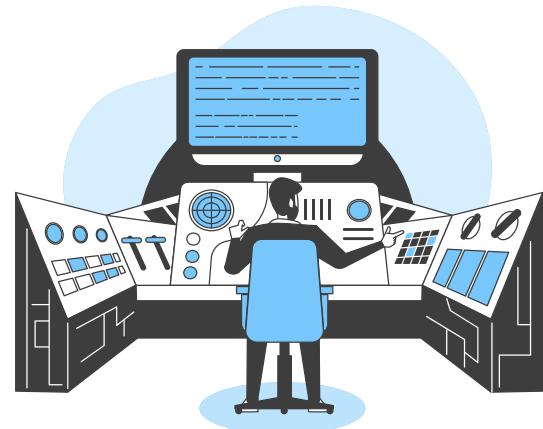
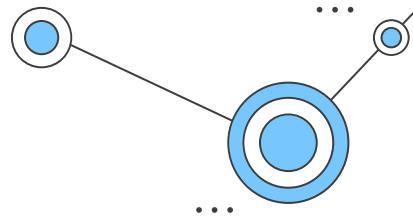
Uma estrutura de dados é um relacionamento lógico entre diferentes tipos de dados visando a resolução de determinado problema de forma eficiente.



1.1 Introdução a algoritmos e estruturas de dados

As **estruturas de dados** são utilizadas em diversas áreas do conhecimento e podem possuir diferentes propósitos.

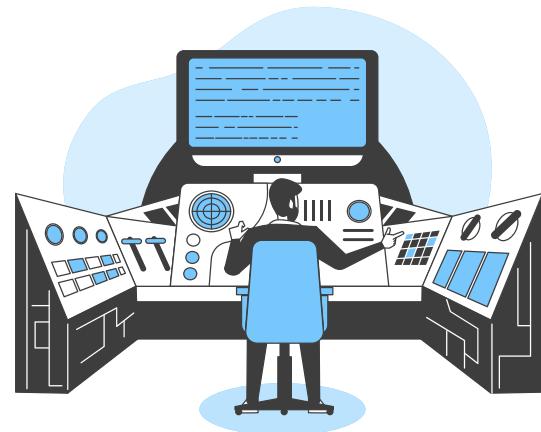
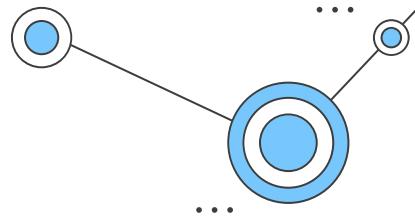
Trata-se de um tema fundamental da ciência da computação, pois a organização de forma coerente dos dados permite a diminuição do custo de execução de um algoritmo em termos de **tempo de execução, consumo de memória** ou em ambos.



1.1 Introdução a algoritmos e estruturas de dados

Uma estrutura de dados é uma forma eficiente de armazenar e organizar os dados dentro do computador para que nossos algoritmos tirem o melhor proveito deles.

Aplicadas em diversas áreas, as estruturas de dados são os fundamentos da Ciência da Computação, uma vez que definem relacionamentos lógicos entre vários tipos de dados, com o objetivo de resolver problemas com eficiência, além de permitirem organizar dados com coerência e aumentar o desempenho de um algoritmo.



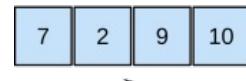
1.1 Introdução a algoritmos e estruturas de dados

Diferentes tipos de estruturas são adequados a diversos tipos de aplicações, sendo algumas altamente especializadas para tarefas específicas.

Exemplos de estruturas de dados:

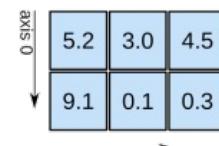
- Arrays;
- Listas encadeadas (ou ligadas);
- Filas e Pilhas;
- Árvores;
- Tabelas Hash.

1D array



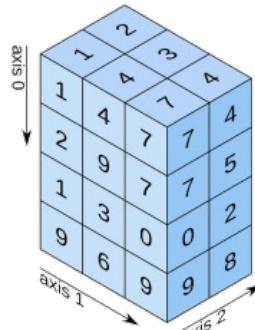
shape: (4,)

2D array



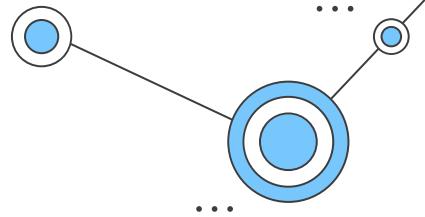
shape: (2, 3)

3D array



shape: (4, 3, 2)

1.1 Introdução a algoritmos e estruturas de dados



Sugestão de leitura:

- <https://meiobit.com/272167/top7-the-greatest-epic-failures-of-the-computing/>
- <https://gizmodo.uol.com.br/hifen-nasa/>
- <https://gizmodo.uol.com.br/exomars-falha-de-software/>

O hífen que destruiu um foguete da NASA

Em 22 de julho de 1962, o Mariner I estava em sua plataforma, pronto para fazer história. Depois do investimento de anos de construção, cálculos e financiamento, a NASA tinha grandes esperanças de que seu foguete realizaria com sucesso um voo de reconhecimento até Vênus, o que daria impulso para a corrida espacial americana. Em [...]



Erro de software pode ter causado a queda da sonda da ExoMars em Marte

Estudos preliminares indicam que uma falha de software causou a queda da sonda Schiaparelli em Marte na semana passada.



AFOGANDO EM NÚMEROS

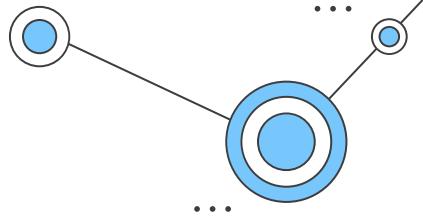
Equipamento da Nasa recebeu informações conflitantes, causando prejuízo de US\$ 125 mi

Conversão errada destruiu sonda espacial

da Reportagem Local

Um problema de conversão de medidas já levou a Nasa (agência espacial norte-americana) a perder uma sonda espacial. A Mars Climate Orbiter foi destruída ao tentar entrar na órbita de Marte em setembro do ano passado. Ao se aproximar do planeta, a sonda recebeu duas informações conflitantes dos controladores na Terra. Uma, no Sistema Métrico Decimal (que usa metro e quilograma), e outra, em unidades britânicas (que usa pé e libra). Os sistemas de engenharia da Nasa não foram capazes de detectar as diferenças numéricas e corrigir os dados, importantes para que a sonda alcançasse a órbita apropriada de Marte, a tempo. O erro fez com que a Mars chegassem 100 km mais perto que o planejado e 25 km abaixo do nível de segurança. A Nasa perdeu US\$ 125 milhões.

1.2 Aplicações de algoritmos

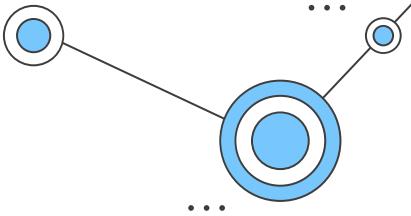


Os algoritmos são fundamentais em muitas áreas da Ciência da Computação e têm uma ampla gama de aplicações em diversos campos.

Vejamos alguns **exemplos reais** de aplicações de algoritmos em diferentes contextos.



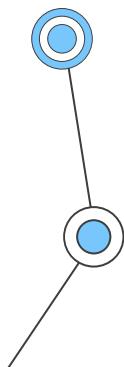
1.2 Aplicações de algoritmos



Ordenação de dados:

Aplicação: Classificação de resultados de pesquisa, organização de dados em bancos de dados, ordenação de elementos em listas.

Exemplo real: A Amazon utiliza algoritmos de ordenação para classificar resultados de pesquisa, apresentar produtos em ordem relevante para o usuário e otimizar a eficiência em seus armazéns.

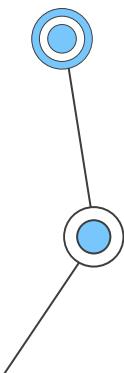
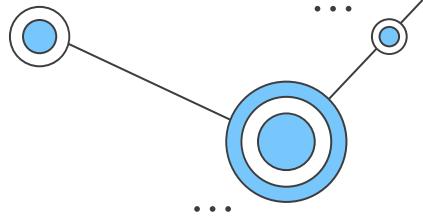


1.2 Aplicações de algoritmos

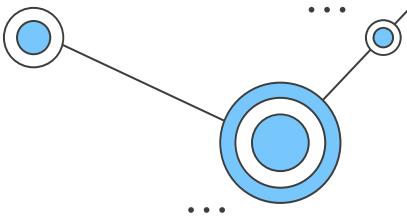
Busca binária:

Aplicação: Busca eficiente em bancos de dados ordenados, localização de itens em listas ou arrays.

Exemplo real: O Google usa busca binária em seus índices para fornecer resultados de pesquisa rápidos e eficientes, localizando informações relevantes em grandes conjuntos de dados.



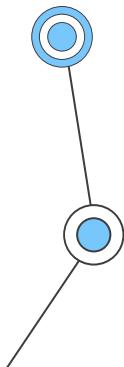
1.2 Aplicações de algoritmos



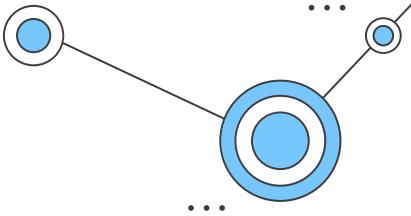
Algoritmos de caminho mínimo:

Aplicação: Roteamento em redes de computadores, navegação em mapas, otimização de rotas para entrega de mercadorias.

Exemplo real: Empresas de transporte/entregas, como a Uber, utilizam algoritmos de caminho mínimo para otimizar as rotas de motoristas, garantindo uma entrega eficiente em termos de tempo e custo.



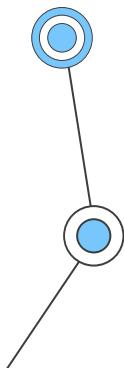
1.2 Aplicações de algoritmos



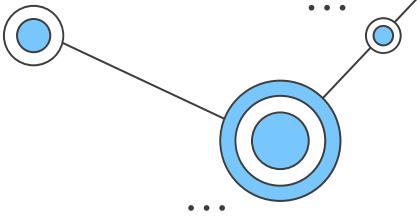
Algoritmos de grafos:

Aplicação: Redes sociais, análise de redes, roteamento em sistemas de transporte público.

Exemplo real: O Facebook utiliza algoritmos de grafos para sugerir conexões entre usuários com base em amizades mútuas, analisando a estrutura de redes sociais.



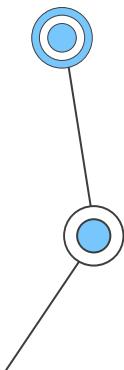
1.2 Aplicações de algoritmos



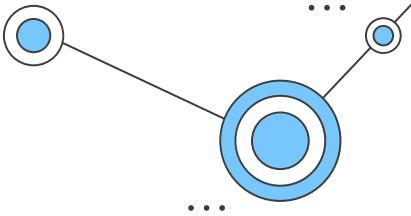
Algoritmos de árvores:

Aplicação: Estruturas de diretórios de sistemas de arquivos, otimização de operações em bancos de dados, representação de hierarquias em organizações.

Exemplo real: Bancos como o Wells Fargo podem usar árvores para organizar a estrutura de diretórios em sistemas de arquivos, facilitando o acesso rápido e eficiente aos dados.



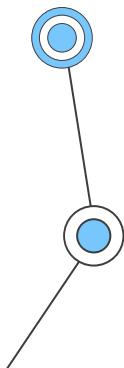
1.2 Aplicações de algoritmos



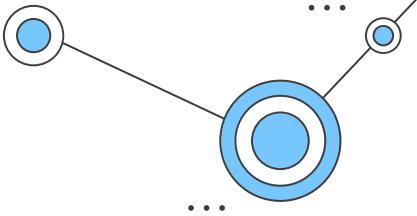
Algoritmos de criptografia:

Aplicação: Segurança em comunicações online, autenticação, proteção de dados sensíveis.

Exemplo real: O WhatsApp aplica algoritmos criptográficos para garantir a privacidade das mensagens, protegendo a comunicação online de usuários.



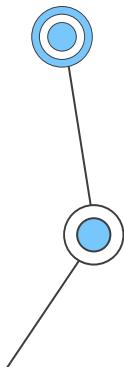
1.2 Aplicações de algoritmos



Algoritmos de compressão de dados:

Aplicação: Compactação de arquivos para economizar espaço de armazenamento, transmissão eficiente de dados pela internet.

Exemplo real: Plataformas de streaming, como o Netflix, utilizam algoritmos de compressão para transmitir vídeos de alta qualidade de maneira eficiente pela internet, economizando largura de banda.

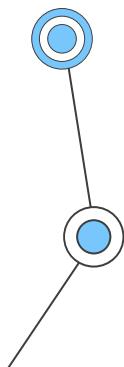
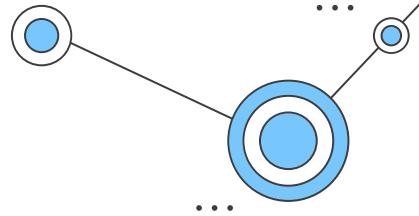


1.2 Aplicações de algoritmos

Algoritmos de aprendizado de máquina:

Aplicação: Classificação de emails como spam ou não spam, recomendações de produtos em plataformas de comércio eletrônico, reconhecimento de padrões em imagens.

Exemplo real: A Netflix utiliza algoritmos de aprendizado de máquina para recomendar filmes e séries com base no histórico de visualização do usuário, melhorando a experiência de usuário.

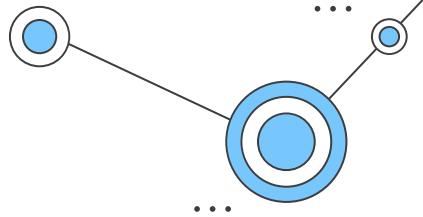


1.2 Aplicações de algoritmos

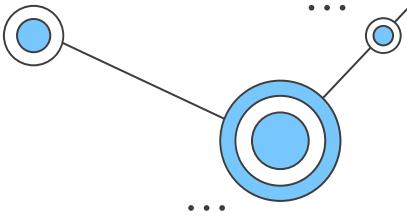
Algoritmos genéticos:

Aplicação: Otimização de processos, design de sistemas, resolução de problemas complexos.

Exemplo real: Empresas de design automotivo, como a Tesla, podem usar algoritmos genéticos para otimizar o design de carros, levando em consideração diversos parâmetros de desempenho.



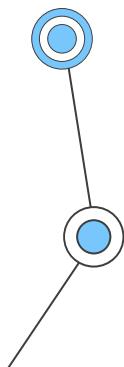
1.2 Aplicações de algoritmos



Algoritmos de processamento de imagens:

Aplicação: Reconhecimento facial, melhoria de qualidade de imagem, segmentação de objetos.

Exemplo real: O Instagram aplica algoritmos de processamento de imagens para reconhecimento facial, aplicação de filtros e melhoria automática da qualidade visual das fotos.

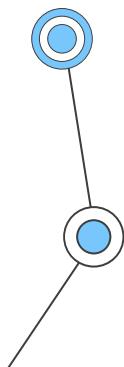
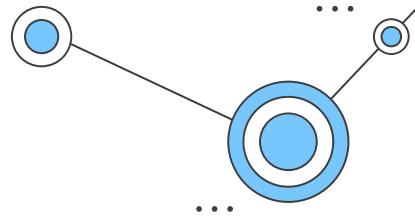


1.2 Aplicações de algoritmos

Algoritmos de roteamento em redes:

Aplicação: Roteamento de pacotes de dados na internet, gerenciamento de tráfego em redes de comunicação.

Exemplo real: Empresas de telecomunicações, como a AT&T, utilizam algoritmos de roteamento para gerenciar o tráfego de dados em suas redes, garantindo uma comunicação eficiente.

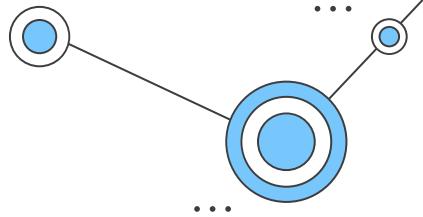


1.2 Aplicações de algoritmos

Algoritmos de otimização:

Aplicação: Otimização de rotas de entrega, planejamento de produção, design de redes de transporte.

Exemplo real: Empresas de logística, como a UPS, empregam algoritmos de otimização para planejar rotas de entrega, minimizando o tempo e os custos envolvidos no transporte de mercadorias.



1.2 Aplicações de algoritmos



HERMES PARDINI
Medicina Diagnóstica e Preventiva

A Empresa Serviços Unidades Assessoria Científica Dúvidas Frequentes Sala de Imprensa Fale Conosco Trabalhe Conosco

buscar >

ANÁLISES CLÍNICAS DIAGNÓSTICO POR IMAGEM PREVENÇÃO DA SAÚDE NÚCLEOS ESPECIALIZADOS

HOME > PROFISSIONAIS DA SAÚDE

ALGORITMOS DE DIAGNÓSTICO MÉDICO

TAMANHO DA FONTE: P+ P-

CERTIFICADOS DE QUALIDADE

Top Hospitalar

O Hermes Pardini foi um dos três finalistas na categoria Medicina Diagnóstica na edição de 2009 do prêmio.



Algoritmo: Abordagem Ecocardiográfica da Endocardite Infecciosa (EI)
Utilizado em: Para Investigação
[▼ DOWNLOAD](#)

Algoritmo: Acidente com Material Biológico
Utilizado em: Para Investigação
[▼ DOWNLOAD](#)

BOLETINS TÉCNICOS

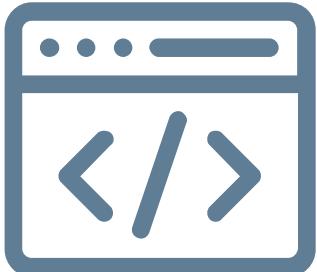
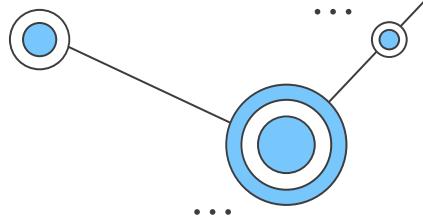
Boletim: Pertussis Disease Burden in the Household -How to Protect Young Infants
Resumo: Estudo para identificar a origem do Bordetella Pertussis (Coqueluche) em crianças de zero e seis meses internados com a doença.
[▼ DOWNLOAD](#)

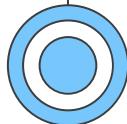
Boletim: Painel Lung Scan NGS
Resumo: Conheça esse serviço, desenvolvido de forma pioneira pelo Laboratório Progenética no Brasil.
[▼ DOWNLOAD](#)

1.3 Modelos, notações e representações

Modelos, notações e representações de algoritmos são conceitos relacionados à descrição e comunicação de algoritmos de maneira mais estruturada e compreensível.

Os modelos oferecem uma visão conceitual, as notações estabelecem uma linguagem padronizada para descrever algoritmos, e as representações são as formas específicas de expressar esses algoritmos, seja visualmente ou por meio de código.





1.3 Modelos, notações e representações

...



Os **modelos de algoritmos** são abstrações de alto nível que fornecem uma visão conceitual e lógica de como um algoritmo funciona.

...

Eles ajudam a entender a lógica subjacente do algoritmo, ignorando detalhes de implementação.

•

Exemplos de modelos incluem pseudocódigo, diagramas de fluxo, linguagens de modelagem como UML (Unified Modeling Language) e descrições verbais detalhadas.

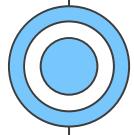
...

•

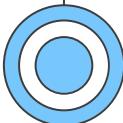
•



...



...



1.3 Modelos, notações e representações

Exemplos de pseudocódigo:
(em alto nível)

```
Algoritmo Media
Var N1, N2, Media : real
Início
    Leia N1, N2
    Media  $\leftarrow$  (N1+N2)/2
    Se Media  $\geq$  7 Entao
        Escreva "Aprovado"
    Senao
        Escreva "Reprovado"
    Fim.
```

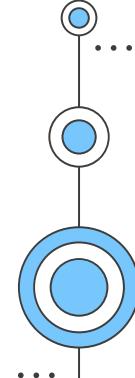
```
INÍCIO
    INTEIRO i, x, y;
    x  $\leftarrow$  1;
    y  $\leftarrow$  1;

    PARA i  $\leftarrow$  ATÉ X  $\leq$  10 FAÇA
        ESCREVA i;

    ENQUANTO x  $\leq$  10 FAÇA
        INÍCIO
            ESCREVA x;
            x  $\leftarrow$  x + 1;
        FIM

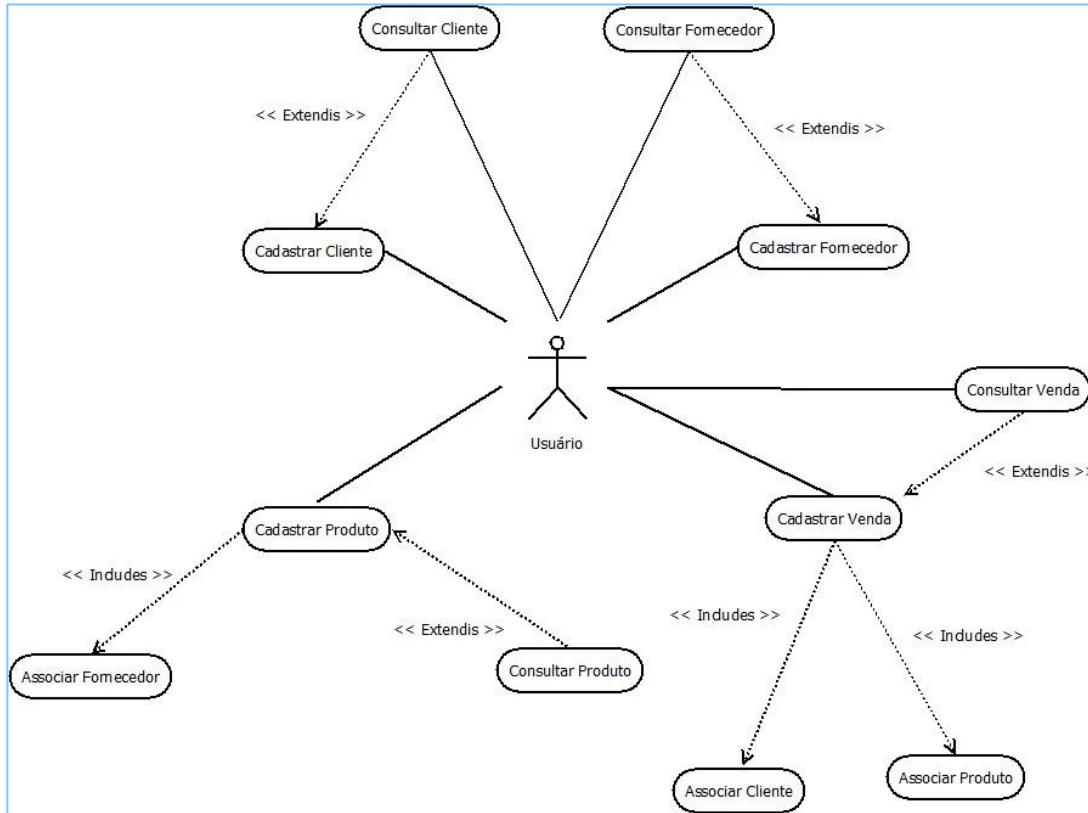
    REPITA
        ESCREVA y;
        y  $\leftarrow$  y + 1;
    ATÉ x  $\leq$  10
    FIM
```

...



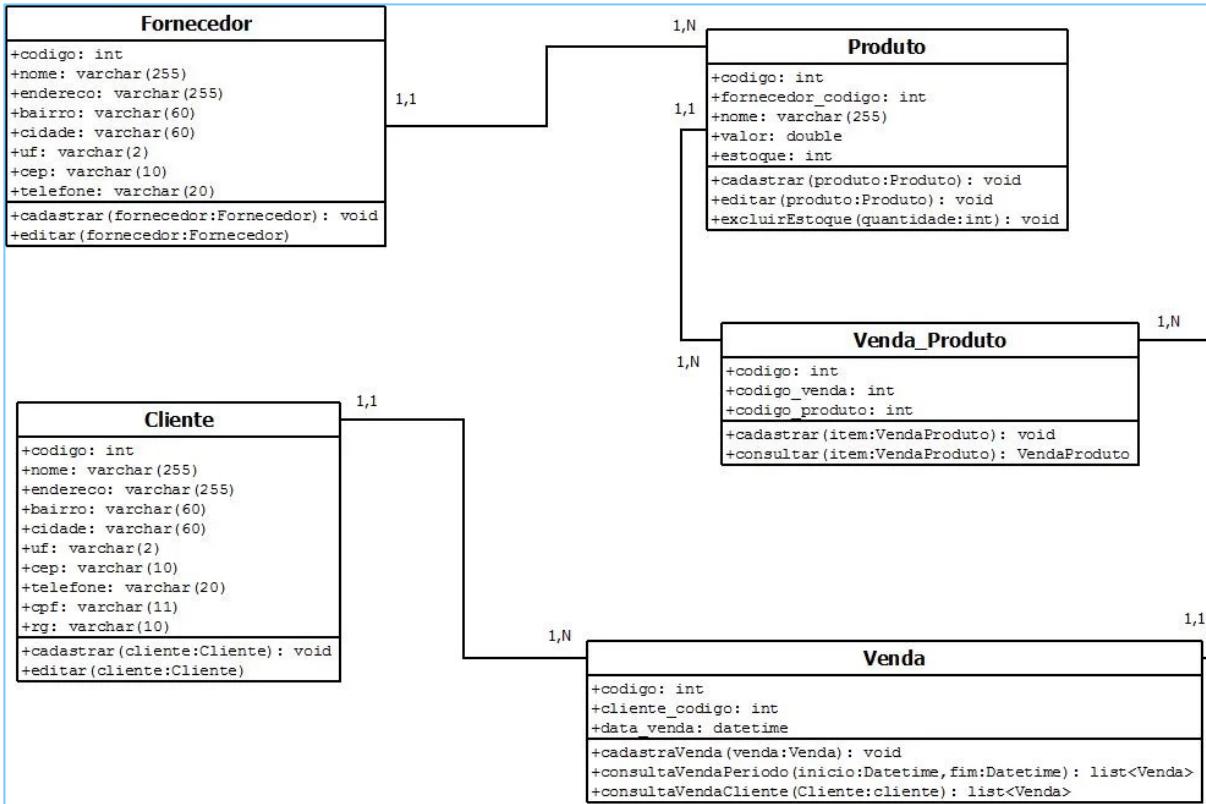
1.3 Modelos, notações e representações

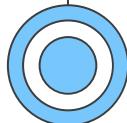
Exemplos da UML (Unified Modeling Language): **Caso de uso:**



1.3 Modelos, notações e representações

Exemplos da UML (Unified Modeling Language): **Diagrama de classe:**





1.3 Modelos, notações e representações

...



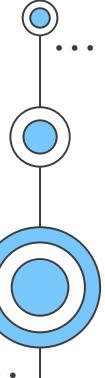
Notações são sistemas de símbolos e regras que fornecem uma maneira padronizada de representar algoritmos. Elas são usadas para expressar a estrutura e o funcionamento dos algoritmos de uma forma mais concisa e compreensível.

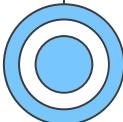
•

•

Pseudocódigo, por exemplo, é uma notação comum usada para descrever algoritmos de maneira mais informal e próxima de uma linguagem de programação real.

...





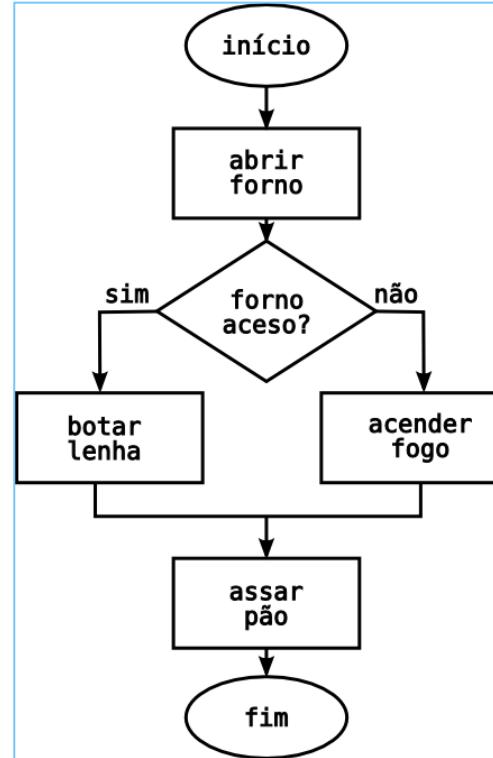
...

1.3 Modelos, notações e representações

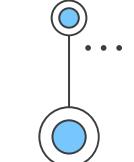
Representações referem-se às diferentes formas visuais ou textuais usadas para apresentar algoritmos.

Diagramas de fluxo são uma representação visual comum, onde diferentes formas geométricas representam diferentes etapas do algoritmo e setas indicam o fluxo de controle.

...



...



...

...



1.3 Modelos, notações e representações



Trechos de código em uma linguagem de programação específica também são representações de algoritmos.

Em C:

```
printf("Hello, World!\n");
```

Em Java:

```
System.out.println("Hello, World!");
```

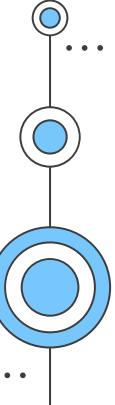
Em Python:

```
print("Hello, World!")
```

Em Portugol:

```
escreva("Hello, World!")
```

...

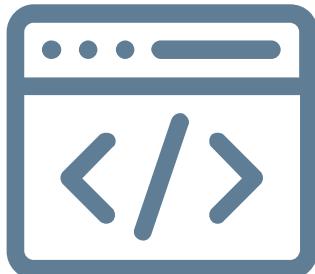
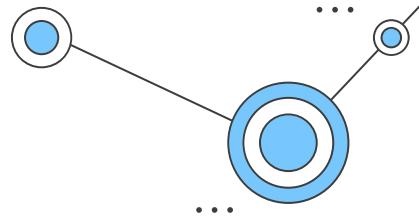


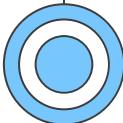
...

1.4 Noções de linguagens de programação

Linguagens de programação são notações para se descrever computações para pessoas e máquinas.

O mundo conforme o conhecemos depende de linguagens de programação, pois todo o software executando em todos os computadores foi escrito em alguma linguagem de programação.





1.4 Noções de linguagens de programação

...
...

Antes que possa rodar, um programa primeiro precisa ser traduzido para um **formato** que lhe permita ser executado por um computador.



Os sistemas de software que fazem essa tradução são denominados compiladores.

```
#include<stdio.h>

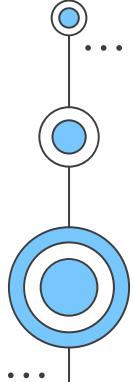
int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

hello_world.c



```
01100110001000100110000111
1100000001111111100000001
1111000110101010001100011
001100010001001100011110
000000111111110000001110
100011010101000110011001
1000100010011000111100000
00111111111000001111100
0110101010001100011001100
0100010011000111110000000
1111111111000001111100011
```

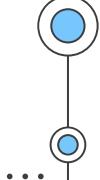
hello_world.o





...

1.4 Noções de linguagens de programação



O estudo da escrita de compiladores é interdisciplinar.

Abrange linguagens de programação, arquitetura de máquina, teoria de linguagem, algoritmos e engenharia de software.

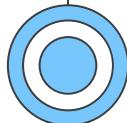
Exemplos de linguagens compiladas: C, C++, C#, Java...

Leitura sugerida:

- <https://github.com/joaopauloaramuni/compiladores/tree/main/PDF>

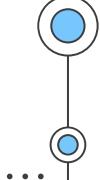


...



...

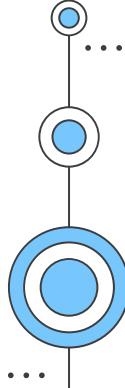
1.4 Noções de linguagens de programação

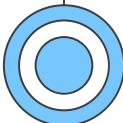


Colocando de uma forma bem simples, um compilador é um programa que recebe como entrada um programa em uma linguagem de programação - a linguagem **fonte** - e o traduz para um programa equivalente em outra linguagem - a linguagem **objeto**.

Um papel importante do compilador é relatar quaisquer erros no programa fonte detectados durante esse processo de tradução.

...





1.4 Noções de linguagens de programação

...
...

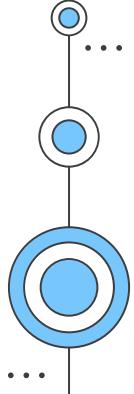
Se o programa objeto for um programa em uma linguagem de máquina executável (.exe por exemplo), poderá ser chamado para processar entradas e produzir saída.



Exemplos de traduções:

- Arquivo .c -> .o
- Arquivo .cpp -> .c
- Arquivo .java -> .class

...





1.4 Noções de linguagens de programação

...

Um **interpretador** é outro tipo comum de processador de linguagem.

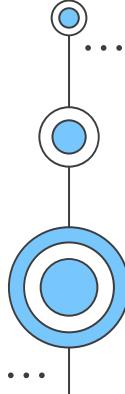
...

Em vez de produzir um programa objeto como resultado da tradução, um interpretador executa diretamente as operações especificadas no programa fonte sobre as entradas fornecidas pelo usuário.

•

Exemplos de linguagens interpretadas: Python, ShellScript, Javascript...

...





...

1.4 Noções de linguagens de programação

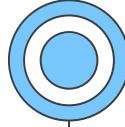


O programa objeto em linguagem de máquina produzido por um compilador normalmente é muito mais rápido no mapeamento das entradas para saídas do que um interpretador.



Porém, um interpretador frequentemente oferece um melhor diagnóstico de erro do que um compilador, pois executa o programa fonte instrução por instrução.

...



...



...



...





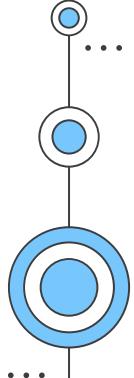
1.4 Noções de linguagens de programação

...
...

Em um **interpretador**, o programa conversor recebe a primeira instrução do programa fonte, confere para ver se está escrita corretamente, converte-a em linguagem de máquina e então ordena ao computador que execute esta instrução.

Depois recebe o processo para a segunda instrução, e assim sucessivamente, até a última instrução do programa fonte. Quando a segunda instrução é trabalhada, a primeira é perdida, isto é, apenas uma instrução fica na memória em cada instante.

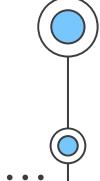
...





...

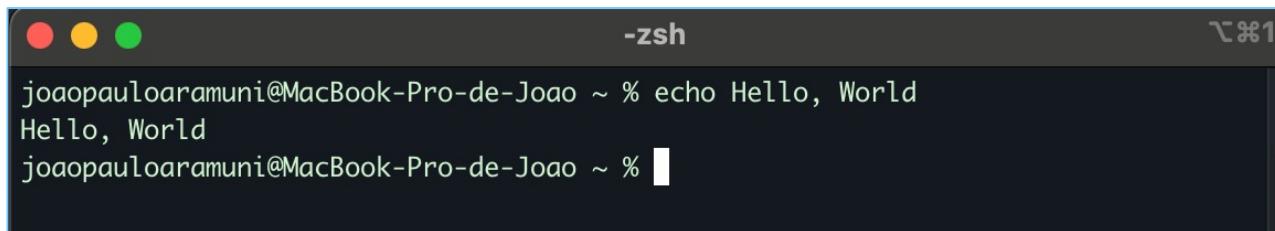
1.4 Noções de linguagens de programação



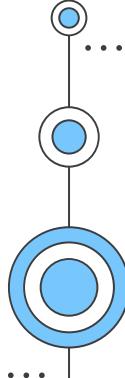
Se este programa for executado uma segunda vez, novamente haverá uma nova tradução, comando por comando, pois os comandos em linguagem de máquina **não** ficam armazenados para futuras execuções.



Neste método, o programa conversor recebe o nome de **interpretador**.



```
joaopauloaramuni@MacBook-Pro-de-Joao ~ % echo Hello, World
Hello, World
joaopauloaramuni@MacBook-Pro-de-Joao ~ %
```

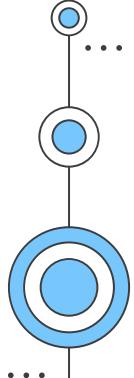


...



1.4 Noções de linguagens de programação

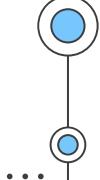
Nos **compiladores**, o programa conversor recebe a primeira instrução do programa fonte, a confere para ver se está escrita corretamente, a converte para linguagem de máquina em caso afirmativo e passa para a próxima instrução, repetindo o processo sucessivamente até a última instrução do programa fonte.





...

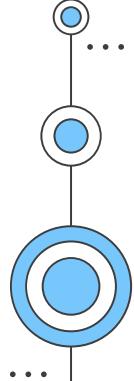
1.4 Noções de linguagens de programação



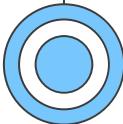
Caso tenha terminado a transformação da última instrução do programa fonte e nenhum erro tenha sido detectado, o computador volta à primeira instrução, já transformada para linguagem de máquina e a executa. Passa à instrução seguinte, a executa, etc., até a última.

Se este programa for executado uma segunda vez, não haverá necessidade de uma nova tradução, uma vez que todos os comandos em linguagem binária foram memorizados em um novo programa completo.

...



...



...

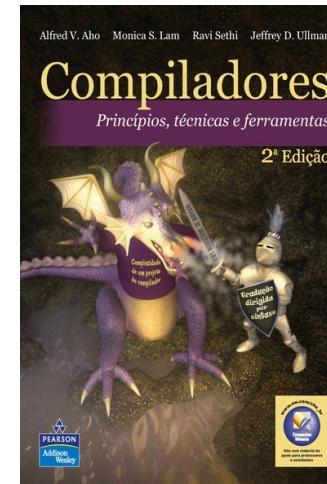
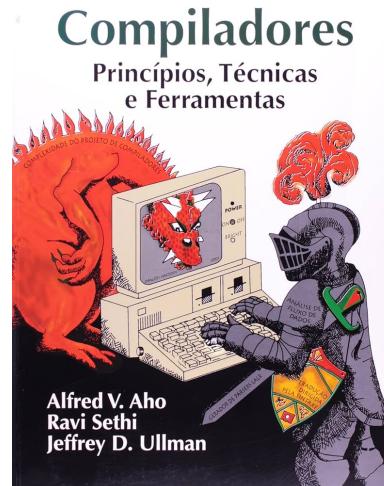
1.4 Noções de linguagens de programação

Neste método, o programa conversor recebe o nome de **compilador**.

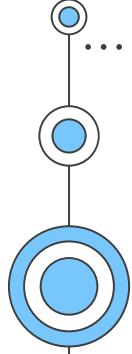
...

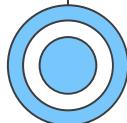
Fonte: Dragon Book.

•



•





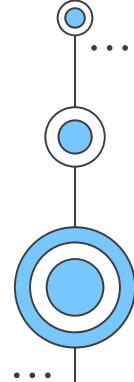
1.4 Noções de linguagens de programação

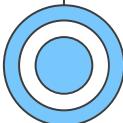
...
...

Vantagem: No processo de compilação a execução fica mais rápida em relação à interpretação, pois se economiza o tempo de retradução de cada instrução a cada nova execução.

Desvantagem: A cada modificação introduzida no programa fonte é necessária uma nova tradução completa para obter um novo programa objeto, o que torna o processo mais difícil na fase de desenvolvimento quando muitas modificações são feitas.

...





1.4 Noções de linguagens de programação

...
...

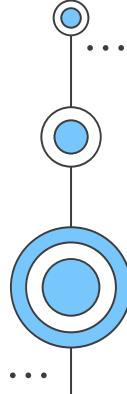
Um compilador que traduz linguagem de alto nível para outra linguagem de alto nível é chamado de tradutor de fonte para fonte ou conversor de linguagem.

...

Um programa que traduz uma linguagem de programação de baixo nível para uma linguagem de programação de alto nível é um **descompilador**.

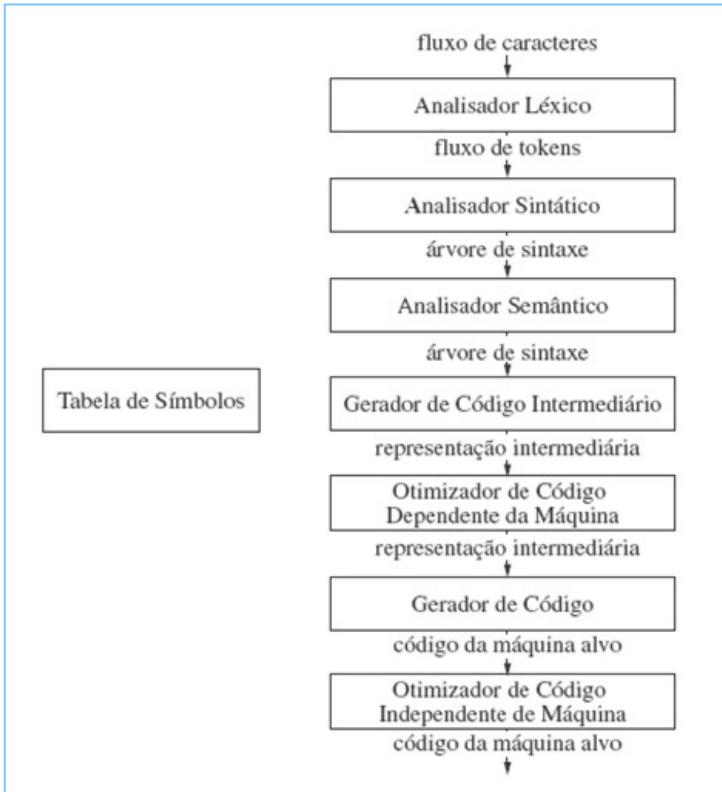
- Exemplo: <https://java-decompiler.github.io/>

...



1.4 Noções de linguagens de programação

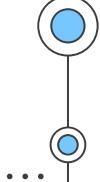
Estrutura básica de um compilador:



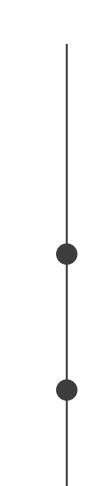


...

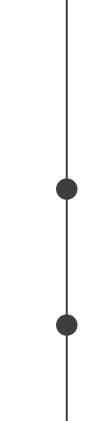
1.4 Noções de linguagens de programação



A fase de **análise** de um compilador subdivide um programa fonte em partes constituintes e produz uma representação interna para ele, chamada código intermediário.

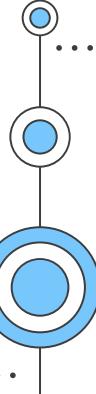


A fase de **síntese** traduz o código intermediário para o programa objeto.



A análise é organizada em torno da “sintaxe” da linguagem a ser compilada. A **sintaxe** de um linguagem de programação descreve a forma apropriada dos seus programas, enquanto a **semântica** da linguagem define o que seus programas significam; ou seja, o que cada programa faz quando é executado.

...



...

1.4 Noções de linguagens de programação

GCC: <https://gcc.gnu.org/>

A GNU Compiler Collection, comumente conhecida como GCC, é um conjunto de compiladores de código aberto desenvolvido pelo Projeto GNU. O GCC suporta várias linguagens de programação, incluindo C, C++, Fortran, Objective-C, e outras.



1.4 Noções de linguagens de programação

GNU: <https://www.gnu.org/>

GNU é um sistema operacional tipo Unix cujo objetivo desde sua concepção é oferecer um sistema operacional completo e totalmente composto por software livre - isto é, que respeita a liberdade dos usuários.



1.4 Noções de linguagens de programação

MinGW (Minimalist GNU for Windows): <https://www.mingw-w64.org/>

O MinGW é uma implementação do GCC para Windows. O MinGW é um projeto que oferece um conjunto de ferramentas de desenvolvimento e bibliotecas para compilar e executar programas nativos do GNU no ambiente Windows.



1.4 Noções de linguagens de programação

MacOSX GCC:

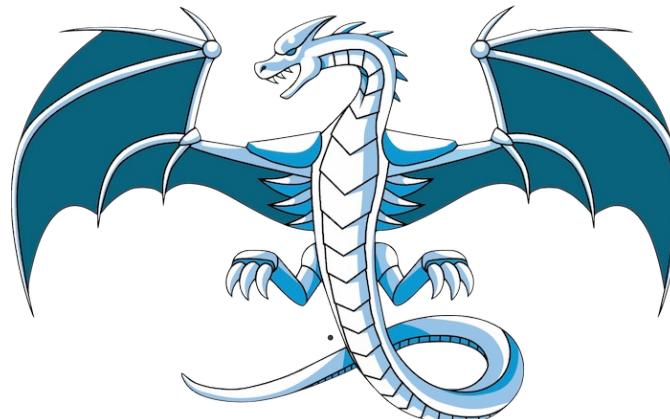
O GCC também está disponível para o macOS, que é o sistema operacional da Apple. Historicamente, o GCC era o compilador padrão no macOS. No entanto, a Apple começou a transição para o uso do Clang como seu compilador padrão a partir do macOS 10.8 (Mountain Lion).

O GCC ainda pode ser instalado no macOS, mas a preferência da Apple pelo Clang é evidente, devido às suas mensagens de erro mais amigáveis e à sua modularidade.

1.4 Noções de linguagens de programação

Clang: a C language family frontend for LLVM: <https://clang.llvm.org/>

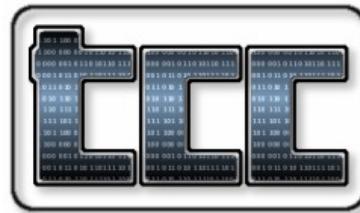
O Clang é um compilador de código aberto desenvolvido pelo projeto LLVM (Low Level Virtual Machine: <https://llvm.org/>). Ele é projetado para ser um substituto moderno e eficiente para o compilador GCC (GNU Compiler Collection) em muitos ambientes.



1.4 Noções de linguagens de programação

Tiny C Compiler: <https://bellard.org/tcc/>

O TinyCC é um compilador C extremamente leve e de código aberto. Ele é conhecido por sua simplicidade e capacidade de compilar rapidamente. O TCC foi projetado com ênfase na simplicidade e na velocidade de compilação. Ele é notavelmente rápido na geração de código executável a partir de código-fonte C.

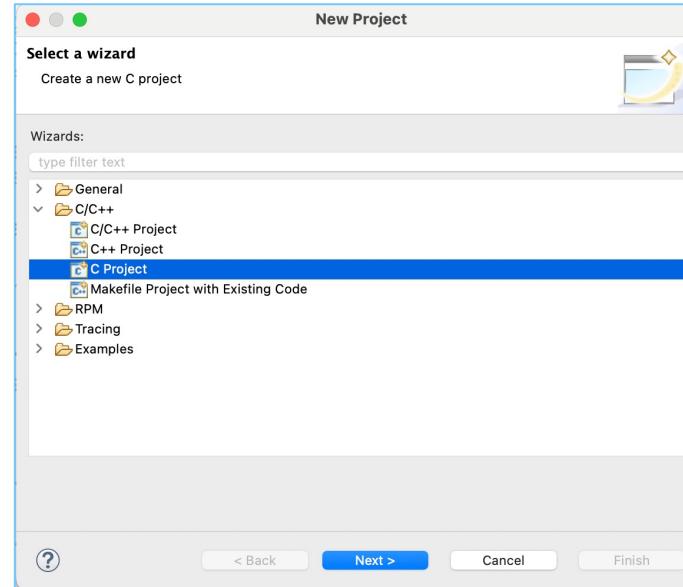
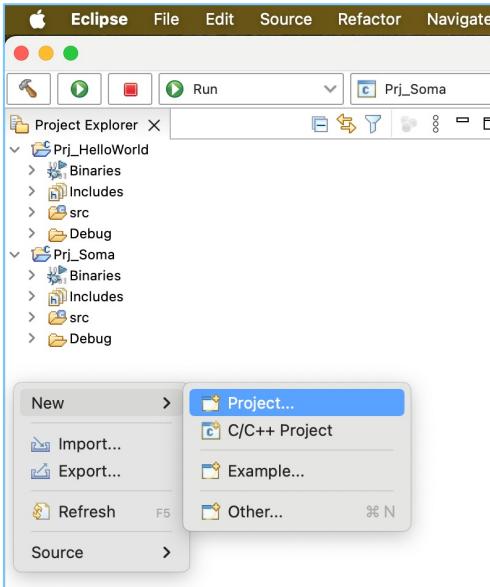


...

1.4 Noções de linguagens de programação

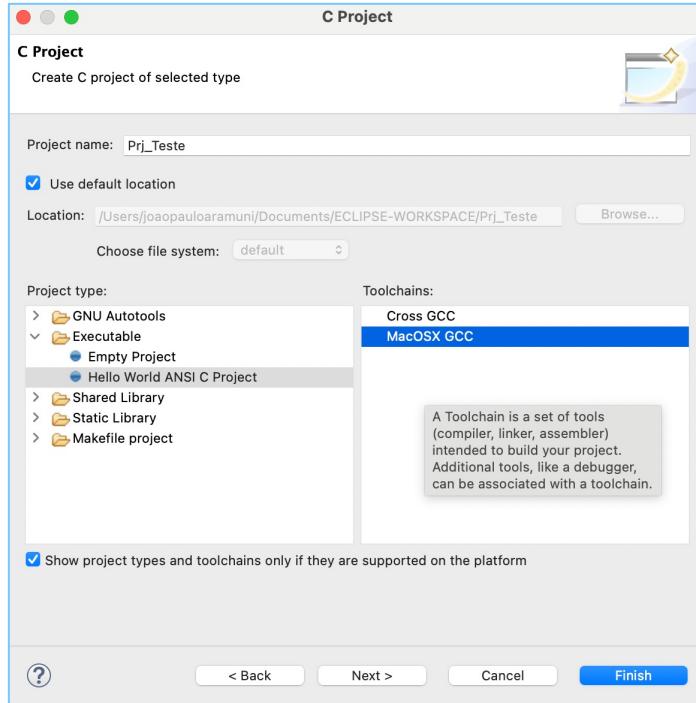
Vamos criar um primeiro projeto para configurarmos o compilador C:

- 1) Baixe e instale o Eclipse para C/C++: Link [aqui](#)
- 2) Crie um novo projeto C.



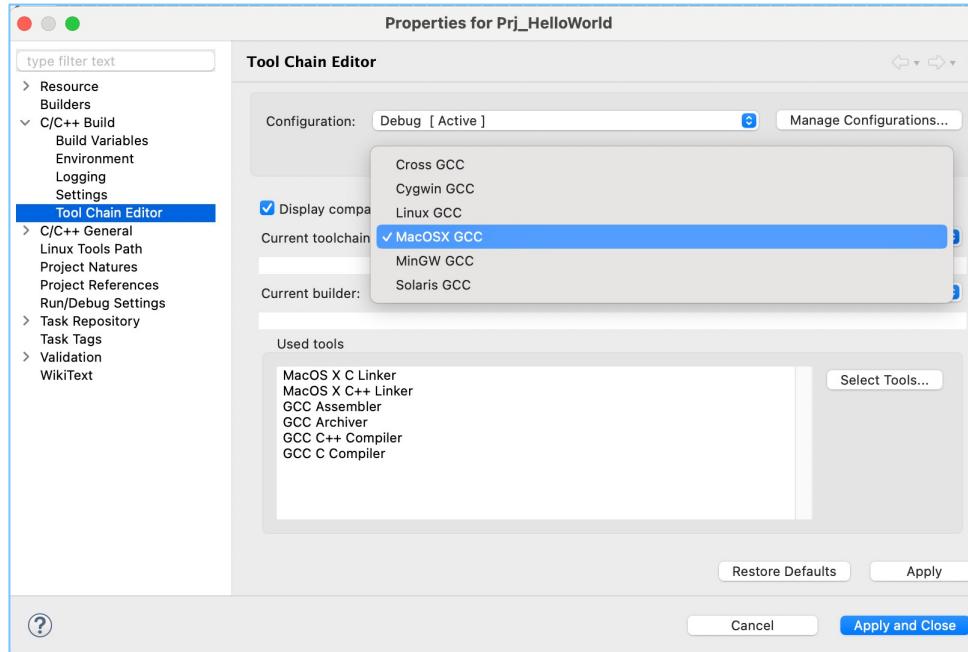
1.4 Noções de linguagens de programação

Como estou no MacOS, vou selecionar o MacOSX GCC. Caso esteja no Windows, selecione o MinGW e caso esteja no Linux, selecione o Linux GCC.



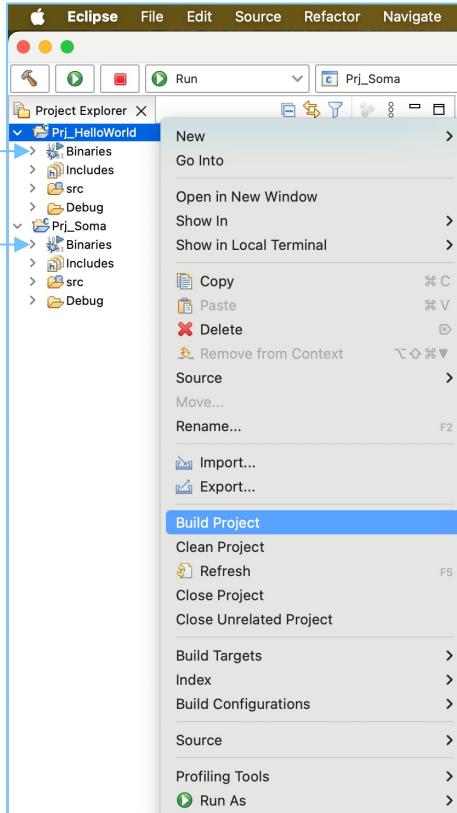
1.4 Noções de linguagens de programação

Para ver qual compilador está selecionado no momento, clique com o botão direito no projeto, vá em propriedades > C/C++ Build > Tool Chain Editor > Current toolchain.



1.4 Noções de linguagens de programação

Em seguida, faça o build do projeto para compilar e construir os arquivos binários.

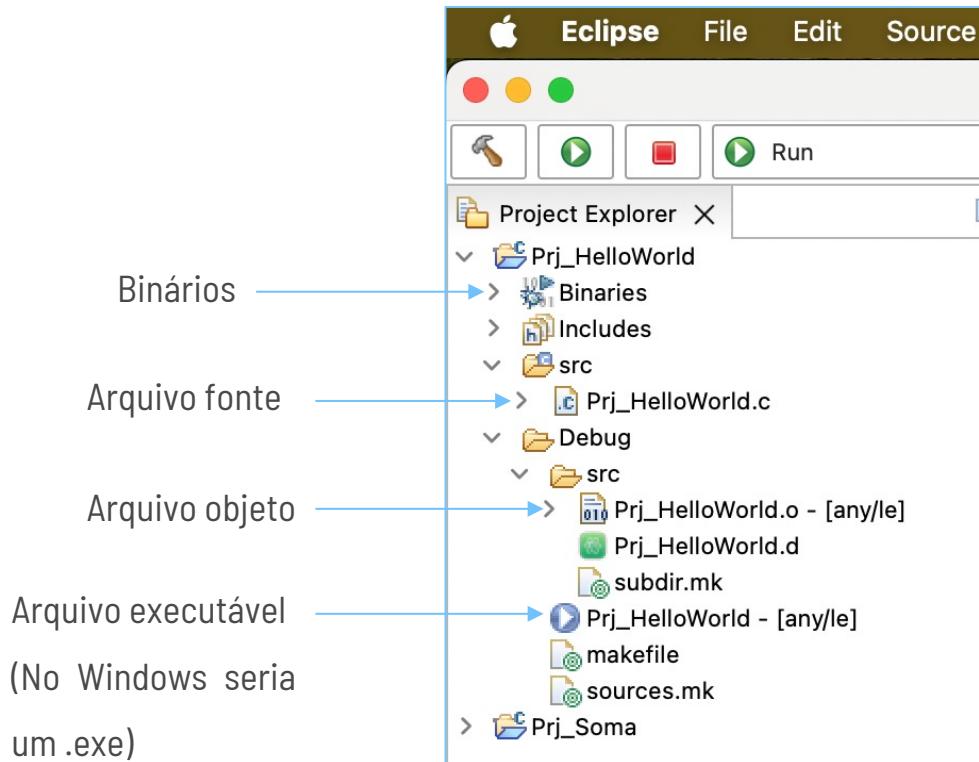


```
CDT Build Console [Prj_HelloWorld]
Building target: Prj_HelloWorld
Invoking: MacOS X C Linker
gcc -o "Prj_HelloWorld" ./src/Prj_HelloWorld.o
Finished building target: Prj_HelloWorld

19:44:12 Build Finished. 0 errors, 0 warnings. (took 146ms)
```

1.4 Noções de linguagens de programação

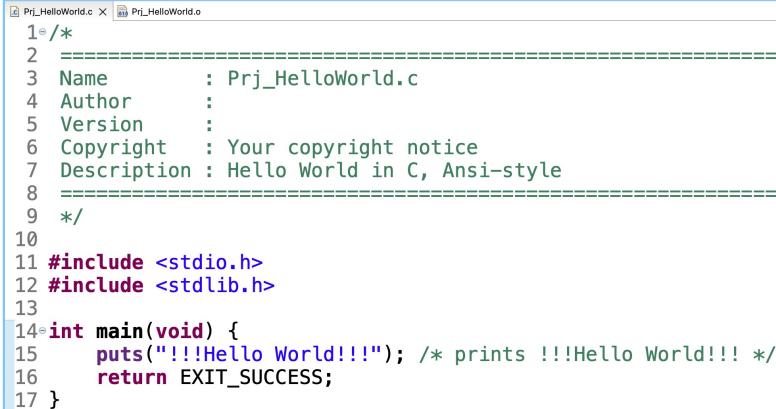
Vejamos como ficou:



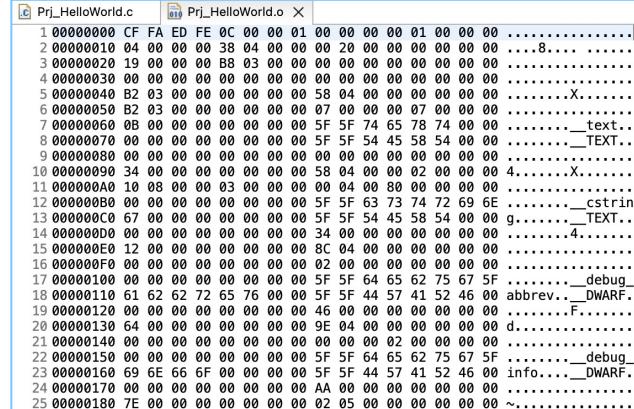
1.4 Noções de linguagens de programação

Arquivo objeto de baixo nível:

.C -> .O



```
1 /*  
2  ======  
3  Name      : Prj_HelloWorld.c  
4  Author    :  
5  Version   :  
6  Copyright : Your copyright notice  
7  Description : Hello World in C, Ansi-style  
8  ======  
9 */  
10  
11 #include <stdio.h>  
12 #include <stdlib.h>  
13  
14 int main(void) {  
15     puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */  
16     return EXIT_SUCCESS;  
17 }
```



```
1 00000000 CF FA ED FE 0C 00 00 01 00 00 00 00 01 00 00 00 .....I  
2 00000010 04 00 00 00 38 04 00 00 00 20 00 00 00 00 00 00 .....8.  
3 00000020 19 00 00 00 B8 03 00 00 00 00 00 00 00 00 00 00 .....  
4 00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
5 00000040 B2 03 00 00 00 00 00 58 04 00 00 00 00 00 00 .....X.....  
6 00000050 B2 03 00 00 00 00 00 00 07 00 00 00 07 00 00 .....  
7 00000060 0B 00 00 00 00 00 00 00 5F 5F 74 65 78 74 00 00 ....._text.  
8 00000070 00 00 00 00 00 00 00 00 5F 5F 54 45 58 54 00 00 ....._TEXT.  
9 00000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....  
10 00000090 34 00 00 00 00 00 00 00 58 04 00 00 02 00 00 00 4.....X..  
11 000000A0 10 08 00 00 03 00 00 00 04 00 80 00 00 00 00 00 .....cstrin  
12 000000B0 00 00 00 00 00 00 00 00 5F 5F 63 73 74 72 69 6E ....._cstrin  
13 000000C0 67 00 00 00 00 00 00 00 5F 5F 54 45 58 54 00 00 g....._TEXT.  
14 000000D0 00 00 00 00 00 00 00 00 34 00 00 00 00 00 00 00 .....4.....  
15 000000E0 12 00 00 00 00 00 00 00 8C 04 00 00 00 00 00 00 .....  
16 000000F0 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 .....  
17 00000100 00 00 00 00 00 00 00 00 5F 5F 64 65 62 75 67 5F ....._debug  
18 00000110 61 62 62 72 65 76 00 00 5F 5F 44 57 41 52 46 00 abbrev....._DWARF.  
19 00000120 00 00 00 00 00 00 00 00 46 00 00 00 00 00 00 00 .....F.....  
20 00000130 64 00 00 00 00 00 00 00 9E 04 00 00 00 00 00 00 .....d.....  
21 00000140 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 .....  
22 00000150 00 00 00 00 00 00 00 00 5F 5F 64 65 62 75 67 5F ....._debug  
23 00000160 69 6E 66 6F 00 00 00 00 5F 5F 44 57 41 52 46 00 info....._DWARF.  
24 00000170 00 00 00 00 00 00 00 00 AA 00 00 00 00 00 00 00 .....  
25 00000180 7E 00 00 00 00 00 00 00 02 05 00 00 00 00 00 00 ~.....
```



...

1.4 Noções de linguagens de programação

Após o build, podemos executar a função main. A main() é uma função especial que serve como ponto de entrada para a execução do programa. Cada programa em C deve ter uma função main() para iniciar a execução.

Botão direito > Run As > Local C/C++ Application

The screenshot shows a code editor with a C/C++ file containing a simple 'Hello World' program. Below the editor is a terminal window showing the build process and success message. A context menu is open over the code, with 'Run As' selected. A submenu shows options for 'C/C++ Container Application' and 'Local C/C++ Application', with the latter being the chosen option. The terminal window then displays the program's output: 'Hello World!!!'.

```
14 int main(void) {  
15     puts("!!!Hello World!!!");  
16     return EXIT_SUCCESS;  
17 }  
18
```

Quick Fix
Source
Refactor
Declarations
References
Search Text
Build Selected File(s)
Clean Selected File(s)
Resource Configurations
Run As
Debug As
Profile As
Profiling Tools
GitHub
Team
Compare With
Replace With
Validate
Preferences...

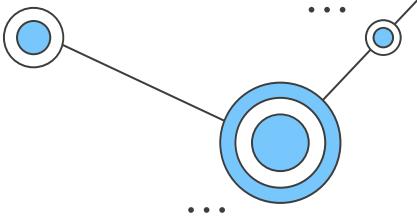
!Hello World!!!

CDT Build Console [Prj_HelloWorld]
Building target: Prj_HelloWorld
Invoking: MacOS X C Linker
gcc -o "Prj_HelloWorld" ./src/Prj_HelloWorld.o
Finished building target: Prj_HelloWorld
19:44:12 Build Finished. 0 errors, 0 warnings. (took 146ms)

1 C/C++ Container Application
2 Local C/C++ Application
Run Configurations...

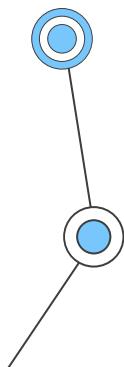
Console X Problems Tasks Properties
<terminated> (exit value: 0) Prj_HelloWorld [C/C++ Application]
!!!Hello World!!!

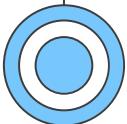
1.5 Conceitos e utilizações



A linguagem C é uma linguagem de programação genérica inventada na década de 1970 por Dennis Ritchie (que também ajudou a criar o Unix).

A influência da linguagem C na história da computação foi significativa, e ela serviu de base para muitas outras linguagens de programação, como C++, Objective-C, C#, Java, Golang, Rust, Swift...





...

1.5 Conceitos e utilizações

- Um programa em C consiste de várias funções encadeadas.
- Uma função é um bloco de código de programa que pode ser usado diversas vezes em sua execução.
- Blocos de código são delimitados por chaves: {}
- O uso de funções permite que o programa fique mais legível, ou seja, mais estruturado.

...



...



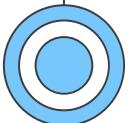
...



...



...



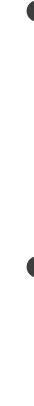
...

1.5 Conceitos e utilizações

...

- O C é “Case Sensitive”:
- Caracteres maiúsculos e minúsculos fazem diferença:
 - `Soma` ≠ `soma` ≠ `SOMA` ≠ `SomA`
- Comandos do C (**if** ou **for**, por exemplo) só podem ser escritos em minúsculas, caso contrário o compilador interpretará como variáveis.

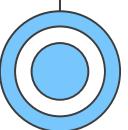
...



...



...



1.5 Conceitos e utilizações

Estrutura geral:

tipo_de_retorno main()

{

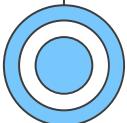
comandos

return valor;

}

...





...

1.5 Conceitos e utilizações

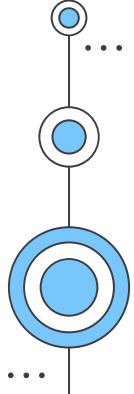
...

Bibliotecas:

Programas que possuem a especificação das funções pré-definidas da linguagem. As bibliotecas que serão usadas devem ser os primeiros comandos de um programa em C.

#include <nome_da_biblioteca.h>

...



...



...

1.5 Conceitos e utilizações

...

stdio.h: biblioteca de funções de entrada e saída (leitura e escrita de dados).

stdlib.h: biblioteca de funções relacionadas a operações de sistema, conversões de tipos, alocação de memória, controle de processos, entre outras.

math.h: biblioteca de funções matemáticas (potenciação, raiz quadrada, funções trigonométricas, etc.).

string.h: biblioteca de funções para manipulação de conjunto de caracteres (por exemplo, palavras). ...



...



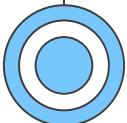
...



...



...



1.5 Conceitos e utilizações

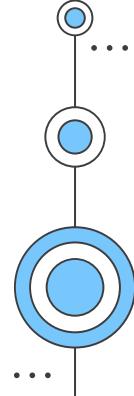
Comentários:

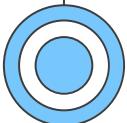
Comentários em C podem ser escritos em qualquer lugar do texto para facilitar a interpretação do algoritmo.

De bloco: `/* Tudo que estiver dentro do bloco será considerado um comentário (e será ignorado pelo compilador). */`

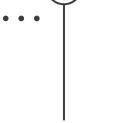
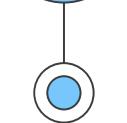
De linha: `// Tudo que estiver à direita será considerado um comentário.`

...





1.5 Conceitos e utilizações

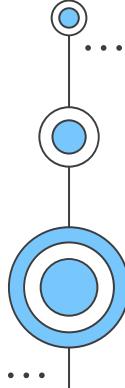


Indentação:

Termo aplicado ao código fonte de um programa para ressaltar ou definir a estrutura do algoritmo.

Na maioria das linguagens de programação, a indentação é empregada com o objetivo de ressaltar a estrutura do algoritmo, aumentando assim a legibilidade do código.

...



1.5 Conceitos e utilizações

Use indentação para destacar blocos de código. Por exemplo, ao definir funções, estruturas de controle de fluxo (if, for, while), e blocos de código em geral.

```
int main() {
    // Início da função main
    if (condition) {
        // Bloco de código dentro de um if
        //
        ...
    } else {
        // Bloco de código dentro de um else
        //
        ...
    }

    for (int i = 0; i < 10; i++) {
        // Bloco de código dentro de um loop for
        //
        ...
    }

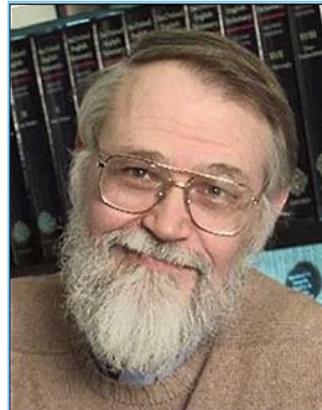
    // Fim da função main
    return 0;
}
```

1.5 Conceitos e utilizações

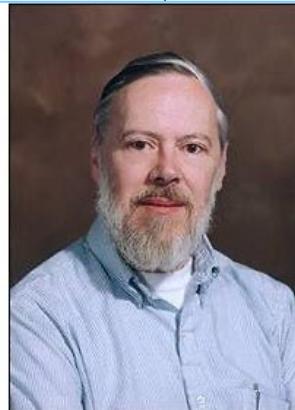
Muitas convenções de estilo em C sugerem colocar a chave de abertura { na mesma linha que a declaração de controle (if, for, while) e a chave de fechamento } alinhada com o início da declaração que a abriu. Isso é conhecido como estilo "K&R" (Kernighan and Ritchie).

```
if (condition) {
    // Bloco de código
} else {
    // Outro bloco de código
}

// Se você fizer assim, deixamos de ser amigos
if (condition)
{
    // Bloco de código
} else
{
    // Outro bloco de código
}
```

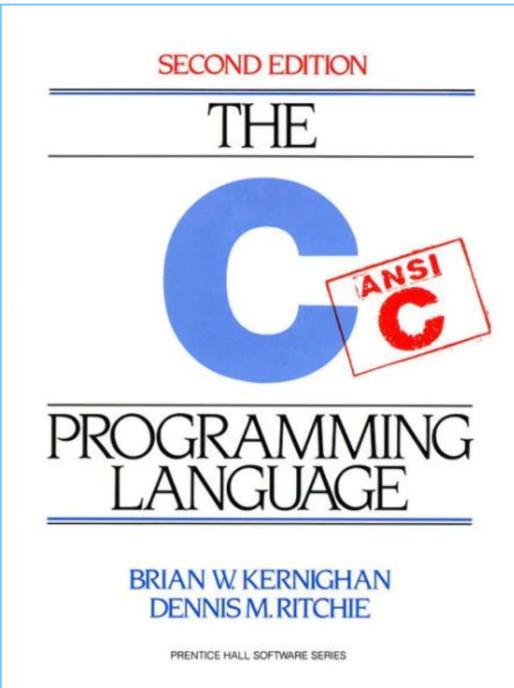


Brian Kernighan



Dennis Ritchie

1.5 Conceitos e utilizações



C Programming Language, 2nd Edition 2nd Edition

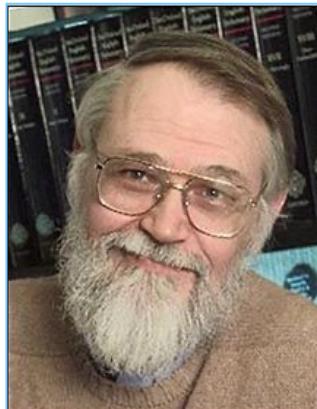
by Brian W. Kernighan (Author), Dennis M. Ritchie (Author)

4.7 ★★★★★ 3,966 ratings

#1 Best Seller in Computer Programming Languages

[See all formats and editions](#)

The authors present the complete guide to ANSI standard C language programming. Written by the developers of C, this new version helps readers keep up with the finalized ANSI standard for C while showing how to take advantage of C's rich set of operators, economy of expression, improved control flow, and data structures. The 2/E has been completely rewritten with additional examples and problem sets to clarify the implementation of difficult language constructs. For years, C programmers have let K&R guide them to building well-structured and efficient programs. Now this same help is available to those working with ANSI compilers. Includes detailed coverage of the C language plus the official C language reference manual for at-a-glance help with syntax notation, declarations, ANSI changes, scope rules, and the list goes on and on.



Brian Kernighan

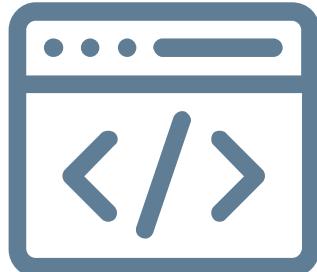
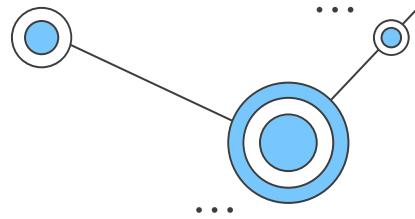


Dennis Ritchie

1.6 Representação de dados

Vejamos agora como os dados são representados na Linguagem C.

Na Linguagem C, a representação de dados é fundamental para a manipulação eficiente da informação. Utilizando variáveis, que são espaços nomeados na memória, os programadores podem armazenar e gerenciar diferentes tipos de dados, como inteiros, ponto flutuante, caracteres e estruturas mais complexas.

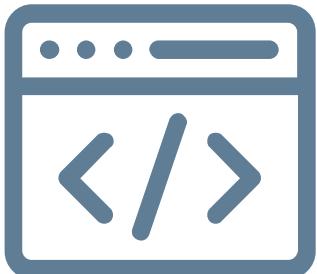
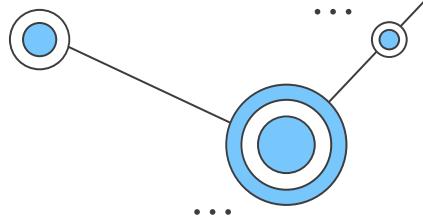


1.6 Representação de dados

Lembre-se: A escolha cuidadosa de tipos de dados e sua representação na memória é crucial para otimizar o desempenho do programa.

A linguagem C oferece uma variedade de tipos e estruturas de dados, permitindo aos desenvolvedores controlar diretamente como os dados são armazenados e acessados.

Mais adiante, veremos em detalhes cada um dos tipos de variáveis existentes.



1.6 Representação de dados

Você deve ter notado o `%d` no exemplo da soma de dois inteiros.

Em C, `%d` é um especificador de formato utilizado em funções de entrada e saída para indicar que você está trabalhando com um argumento ou variável do tipo inteiro (decimal).

```
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 int soma(int a, int b){
15     int c = a + b;
16     return c;
17 }
18
19 int main(void) {
20     puts("!!!Hello World!!!");
21     int x = 2;
22     int y = 3;
23     int z = soma(x,y);
24     printf("Soma = %d\n", z);
25
26     return EXIT_SUCCESS;
27 }
```

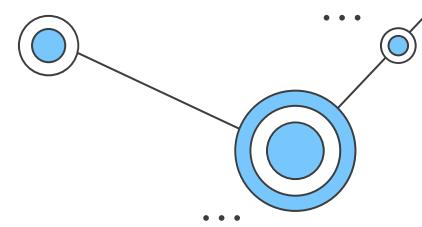
```
Console X Problems Tasks Properties
<terminated> (exit value: 0) Prj_HelloWorld [C/C++ Application] /Users/joaopauloaramuni/Documents/
!!!Hello World!!!
Soma = 5
```

1.6 Representação de dados

Especificadores de formato mais comumente utilizados:

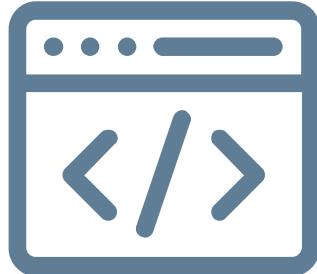
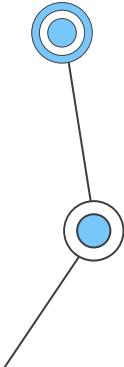
Linguagem C	Formato	Tipo de dados	Exemplo
char	%c	caracter	'A'
int	%d	inteiro	-13 ou 13
int	%u	inteiro sem sinal	13
float	%f	real	3.14
char{}	%s	Cadeia de caracteres (string)	"Aramuni"

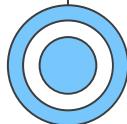
1.7 Boas práticas: noções de documentação, contagem de operações e metodologia para desenvolvimento e testes



Vejamos algumas boas práticas de programação para aplicarmos daqui em diante.

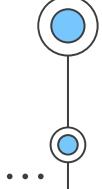
Boas práticas de programação são diretrizes e abordagens recomendadas que os desenvolvedores podem seguir para escrever código de alta qualidade, fácil de entender, manter e colaborar.





...

1.7 Boas práticas em geral

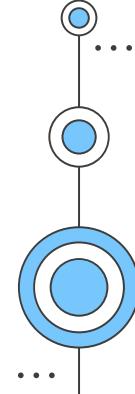


Nomes significativos: Escolha nomes descritivos e significativos para variáveis, funções, classes, etc. Nomes autoexplicativos facilitam a leitura e compreensão do código.



Comentários claros: Use comentários para explicar partes do código que podem não ser óbvias. No entanto, evite comentários óbvios e mantenha-os atualizados à medida que o código evolui.

...

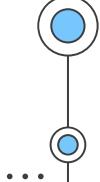


...



...

1.7 Boas práticas em geral



Indentação consistente: Mantenha uma indentação consistente para melhorar a legibilidade do código. A maioria das linguagens utiliza espaços ou tabs, mas o importante é manter a consistência.



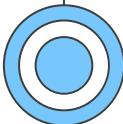
Evite linhas de código muito longas: Linhas de código muito longas podem ser difíceis de ler. Tente manter as linhas dentro de limites razoáveis (geralmente 80-120 caracteres).

...



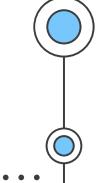
...





...

1.7 Boas práticas em geral



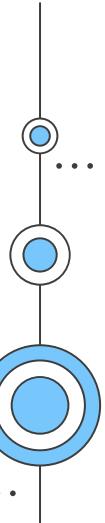
Divisão lógica do código: Divida o código em funções e módulos lógicos.

Cada função deve ter uma responsabilidade clara e específica. Isso facilita a manutenção e reutilização do código.



Evite duplicação de código (DRY - Don't Repeat Yourself): Evite repetir o mesmo bloco de código em vários lugares. Se você precisar fazer uma alteração, a duplicação torna difícil garantir que todas as instâncias sejam atualizadas corretamente.

...

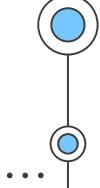


...



...

1.7 Boas práticas em geral

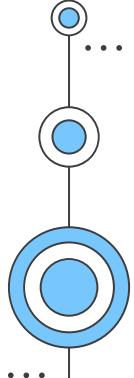


Tratamento de erros adequado: Implemente tratamento de erros adequado para tornar seu código mais robusto. Use exceções (ou mecanismos equivalentes em outras linguagens) para lidar com situações excepcionais.

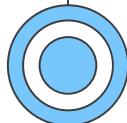


Testes unitários: Escreva testes unitários para validar o comportamento esperado do seu código. Isso ajuda a garantir que as alterações futuras não introduzam regressões.

...

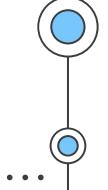


...

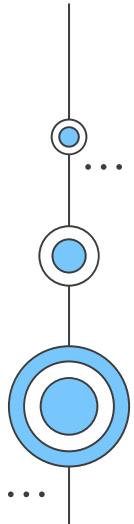


...

1.7 Boas práticas em geral

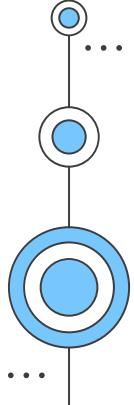


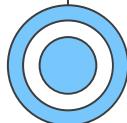
Controle de versão: Utilize sistemas de controle de versão (como Git) para rastrear alterações no código. Isso facilita a colaboração, o rastreamento de alterações e a reversão a versões anteriores se necessário.



Convenções de codificação: Siga as convenções de codificação estabelecidas pela comunidade ou pela equipe. Isso pode incluir estilos de nomenclatura, formatação de código e outras práticas comuns.

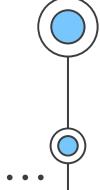
...





...

1.7 Boas práticas em geral

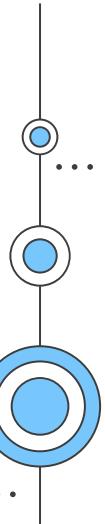


Minimize o acoplamento: Tente minimizar o acoplamento entre diferentes partes do código. Componentes independentes são mais fáceis de manter e reutilizar (leia sobre microserviços, link [aqui](#)).

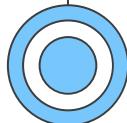


Consciência de desempenho: Esteja ciente das implicações de desempenho do seu código. Escolha algoritmos eficientes e evite práticas que possam impactar negativamente o desempenho.

...

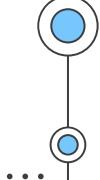


...



...

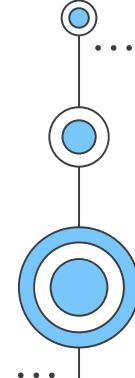
1.7 Boas práticas em geral



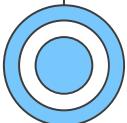
Atualizações incrementais: Faça atualizações incrementais no código, testando e verificando o comportamento a cada passo. Isso facilita a identificação de problemas e a correção antes que se tornem mais complexos.

Documentação adequada: Além de comentários no código, forneça documentação externa (por exemplo, um README) para orientar outros desenvolvedores sobre como usar seu código e configurar o ambiente de desenvolvimento.

...



...



...

1.7 Noções de documentação



...

A **documentação** é uma parte fundamental do processo de desenvolvimento de software, independentemente da linguagem de programação. Ela desempenha um papel crucial na compreensão, manutenção e colaboração do código.



•

•

...

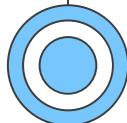


...



...

Vejamos algumas razões para a importância da documentação.



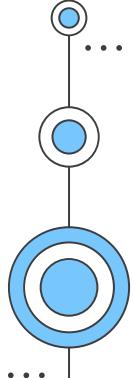
1.7 Noções de documentação

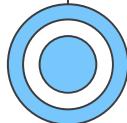
...
...

Compreensão do código: Documentação fornece insights sobre o propósito, funcionamento e lógica por trás do código. Isso é especialmente importante em linguagens de programação de baixo nível, como C, onde a abstração é menor e o código pode ser mais complexo.

Facilita a manutenção: Código bem documentado é mais fácil de manter. Descrições claras ajudam os desenvolvedores a entender a lógica do código, facilitando a identificação e correção de bugs, além de permitir alterações sem causar problemas inesperados.

...

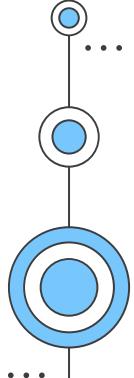




1.7 Noções de documentação

Colaboração eficiente: Em projetos de equipe, a documentação serve como um ponto de referência compartilhado. Novos membros da equipe podem se integrar mais rapidamente ao projeto, e a colaboração é mais eficiente quando todos têm uma compreensão clara do código.

Transparência e boas práticas: Documentação clara promove transparência no desenvolvimento, permitindo que outros desenvolvedores entendam facilmente as escolhas de design, a estrutura do código e as práticas recomendadas.





...

1.7 Noções de documentação



...

Redução da dependência pessoal: Se um desenvolvedor deixa a equipe ou empresa, a documentação torna a transição mais suave. Outros desenvolvedores podem entender e continuar o trabalho sem depender exclusivamente do conhecimento de uma pessoa.



...

Facilita a depuração: Quando ocorrem erros, uma documentação bem elaborada pode ser crucial para entender o contexto e acelerar o processo de depuração.

...



...



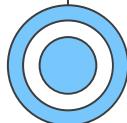
...



...



...



1.7 Noções de documentação

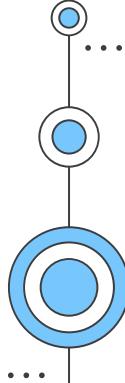
...

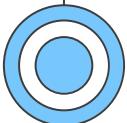
A documentação em C pode incluir diferentes formas de descrições e comentários. Algumas noções de documentação em C incluem:

Comentários de Bloco:

- Use comentários de bloco (com `/* ... */`) para documentar partes significativas do código, como funções, trechos complexos ou cabeçalhos de arquivos. Inclua uma breve descrição do que a seção faz, como ela é usada e, se necessário, informações sobre parâmetros ou retornos.

...





...

1.7 Noções de documentação

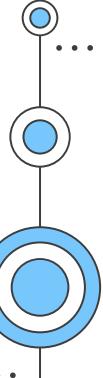


...

Comentários de Bloco:

```
/*
 * Esta função realiza uma tarefa específica.
 * @param parametro1 Descrição do primeiro parâmetro.
 * @param parametro2 Descrição do segundo parâmetro.
 * @return Descrição do valor de retorno.
 */
int minhaFuncao(int parametro1, int parametro2) {
    // Implementação da função
}
```

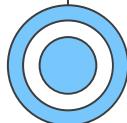
...



...

•

•



...

1.7 Noções de documentação

...

Comentários de Linha:

- Use comentários de linha (com //) para explicar partes específicas do código onde um contexto adicional pode ser útil.

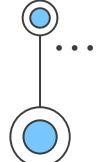
```
int resultado = operacao1() + operacao2(); // Soma o resultado de duas operações
```

Comentários não são a melhor forma de documentação, mas, ainda assim, são uma forma.

...



...



...

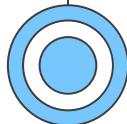
1.7 Noções de documentação

Ferramentas como **Doxygen** geram documentação automaticamente a partir de comentários específicos inseridos no código-fonte, facilitando o trabalho de documentar: <https://www.doxygen.nl/index.html>

The screenshot shows the official Doxygen website at <https://www.doxygen.nl/>. The header includes links for "doxygen", "Docs", "Changelog", "Extensions", and "Examples". A green "Download" button and a "Donate" button are also present. The main content area features a dark blue background with white text. On the left, there's a large, bold text: "Code Documentation. Automated." followed by "Free, open source, cross-platform.". To the right, a search bar and a navigation menu with tabs for "Main Page", "Related Pages", "Namespaces", "Classes", and "Files" are visible. A tooltip for the "find()" function is shown, which reads: "Finds a value in the cache given the corresponding key. Returns a pointer to the value or nullptr if the key is not found in the cache. Note The hit and miss counters are updated, see hits() and misses(). Definition at line 105 of file cache.h." Below this, a code snippet from the file "cache.h" is displayed:

```
166     const_iterator  
167     find(const K& key);  
168     const_iterator  
169     find(const K& key) const;  
170     // move the item to the front of the list  
171     void moveToFront(const K& key);  
172     void moveToFront(const K& key) const;  
173     void moveFront();  
174     void moveFront() const;  
175     void moveBack();  
176     void moveBack() const;  
177     void moveBack(const K& key);  
178     void moveBack(const K& key) const;  
179     void moveBackBy(int n);  
180     void moveBackBy(int n) const;  
181     void moveFrontBy(int n);  
182     void moveFrontBy(int n) const;  
183     void moveFrontBy(int n, const K& key);  
184     void moveFrontBy(int n, const K& key) const;  
185     void moveBackBy(int n, const K& key);  
186     void moveBackBy(int n, const K& key) const;
```

At the bottom of the code block, it says "References Cache< K, V >::m_cachelist, Cache< K, V >::m_cachemap, Cache< K, V >::m_hits, and Cache< K, V >::m_misses." and "Generated by doxygen 1.10.0". A "Show dark mode output" toggle switch is located at the bottom center.

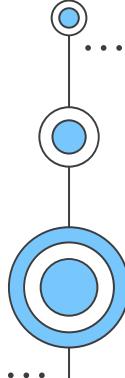


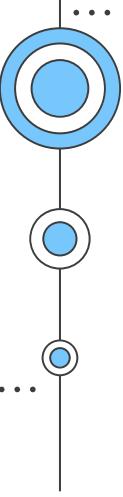
1.7 Contagem de operações

A **contagem de operações** é uma técnica utilizada para analisar o desempenho de algoritmos, especialmente no que diz respeito à sua eficiência em termos de tempo de execução.

O objetivo é estimar o número de operações fundamentais realizadas pelo algoritmo em função do tamanho da entrada. Essa análise fornece uma ideia do crescimento do tempo de execução à medida que o tamanho do problema aumenta.

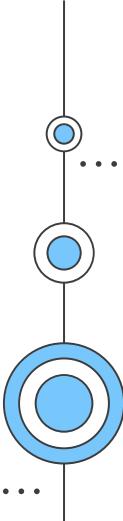
...

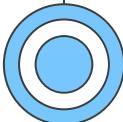




1.7 Contagem de operações

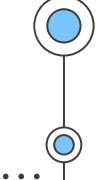
Existem três tipos principais de operações que são comumente contadas durante a análise de algoritmos:

- **Atribuições ou operações básicas:** Refere-se a operações elementares que são executadas em tempo constante, como adições, subtrações, multiplicações, comparações, entre outras.
 - **Comparações:** São operações que envolvem a comparação de dois valores. Em muitos casos, as comparações são operações fundamentais e são cruciais para entender o comportamento de algoritmos de ordenação e busca.
 - **Acesso a dados:** Inclui operações de leitura ou escrita em estruturas de dados, como arrays ou listas. O tempo de acesso a dados pode variar dependendo da estrutura utilizada.
- 



...

1.7 Contagem de operações

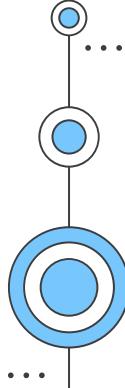


Ao realizar a contagem de operações, é possível classificar os algoritmos em termos de sua **eficiência assintótica**, geralmente expressa usando a notação "O grande" (**Big O**). A notação Big O fornece uma estimativa superior do crescimento do tempo de execução à medida que o tamanho da entrada aumenta.



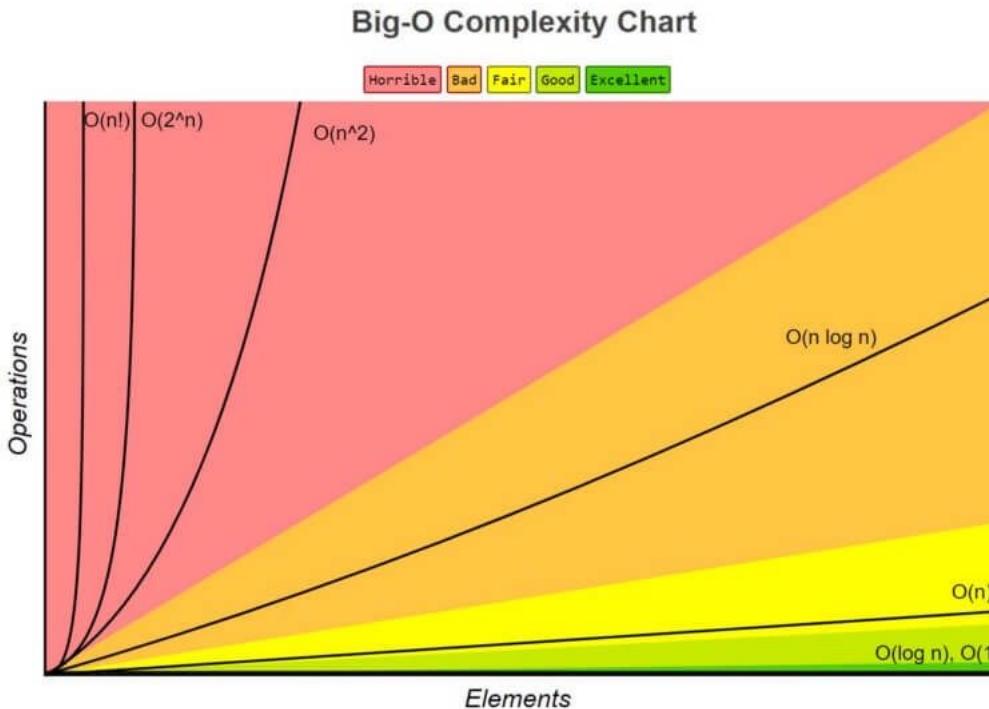
De modo mais geral, o adjetivo assintótico significa "para todos os valores suficientemente grandes."

...



...

1.7 Contagem de operações



1.7 Contagem de operações

Exemplo:

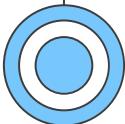
$O(n)$: complexidade linear e;

$O(n^2)$: complexidade quadrática.

```
// Exemplo de loop com complexidade O(n)
void exemplo_on(int n) {
    for (int i = 0; i < n; i++) {
        printf("%d\n", i);
    }
}
```

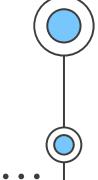
```
// Exemplo de loop com complexidade O(n^2)
void exemplo_on2(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d %d\n", i, j);
        }
    }
}
```

Mais adiante, veremos em detalhes as estruturas de repetição (loops).

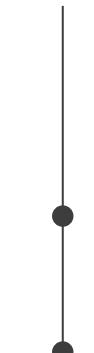


...

1.7 Contagem de operações

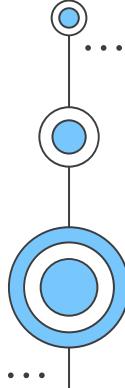


Além disso, é importante notar que a contagem de operações é uma ferramenta teórica e, por vezes, pode não representar com precisão o desempenho real em um ambiente de execução específico.



Fatores como a arquitetura do computador, otimizações de compilador e outros detalhes de implementação também desempenham um papel na eficiência prática de um algoritmo.

...



...



1.7 Metodologia para desenvolvimento e testes

Desenvolvimento **iterativo** vs **incremental**

Iterativo

Entregas

1



2



3



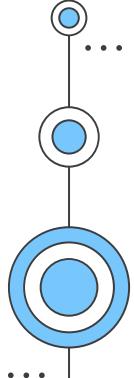
4



5



Incremental





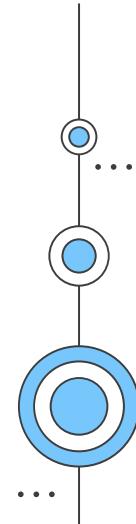
1.7 Metodologia para desenvolvimento e testes

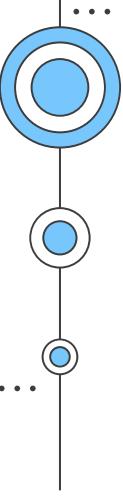
Desenvolvimento **iterativo** (um pouco de tudo)

Ciclos repetidos: O desenvolvimento iterativo divide o projeto em ciclos ou iterações. Cada iteração é uma fase completa do desenvolvimento, incluindo análise, design, implementação e teste.

Feedback contínuo: Cada iteração fornece um incremento funcional ao produto. O feedback é obtido regularmente, permitindo ajustes e adaptações ao longo do tempo.

...





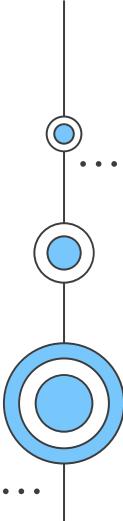
1.7 Metodologia para desenvolvimento e testes

Desenvolvimento **iterativo** (um pouco de tudo)

Adaptação a mudanças: A abordagem iterativa permite a adaptação a mudanças nos requisitos durante o desenvolvimento. Mudanças podem ser incorporadas em iterações subsequentes.

Protótipos: Protótipos podem ser desenvolvidos nas primeiras iterações para validar conceitos e interações com o cliente.

Maior flexibilidade: Oferece flexibilidade para lidar com mudanças e incertezas ao longo do projeto.



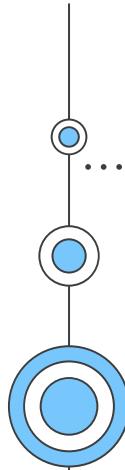


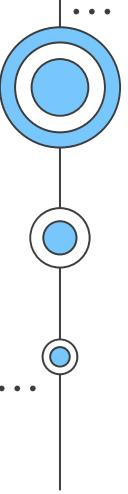
1.7 Metodologia para desenvolvimento e testes

Desenvolvimento **incremental** (tudo de um pouco)

Construção por adição: O desenvolvimento incremental constrói o sistema através da adição sequencial de componentes ou funcionalidades. Cada incremento representa uma parte adicional do sistema.

Entregas frequentes: Entregas são feitas em partes incrementais, aumentando gradualmente a funcionalidade do produto. Cada incremento é uma versão mais completa do sistema.





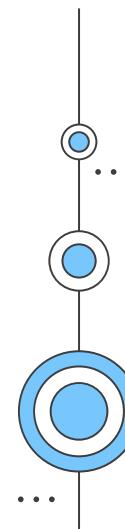
1.7 Metodologia para desenvolvimento e testes

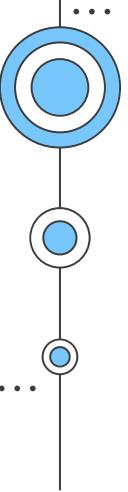
Desenvolvimento **incremental** (tudo de um pouco)

Feedback contínuo: Assim como no desenvolvimento iterativo, o feedback contínuo é obtido após cada entrega incremental.

Visão completa no final: A versão final do sistema é construída através da adição de todos os incrementos planejados. Cada incremento adiciona uma parte significativa à funcionalidade total.

Risco distribuído: O risco é distribuído ao longo do projeto, pois cada incremento é uma entrega completa e testada.





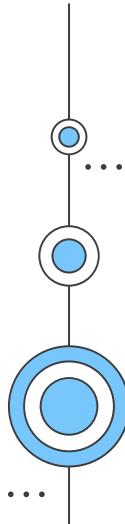
1.7 Metodologia para desenvolvimento e testes

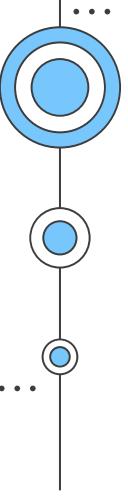
Semelhanças:

Feedback contínuo: Ambas as abordagens enfatizam o feedback contínuo do cliente ou usuário final.

Flexibilidade: Ambas as abordagens oferecem flexibilidade para lidar com mudanças nos requisitos.

Entregas frequentes: Ambas buscam fornecer entregas frequentes e funcionais ao longo do desenvolvimento.



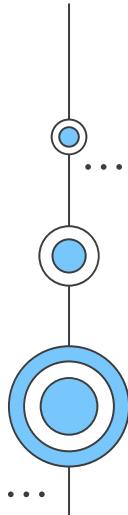


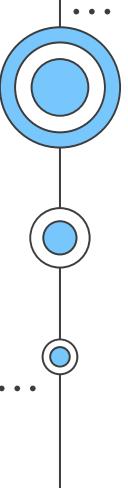
1.7 Metodologia para desenvolvimento e testes

Diferenças:

Estrutura das iterações: O desenvolvimento iterativo divide o projeto em ciclos, enquanto o desenvolvimento incremental constrói o sistema por meio de adições sequenciais.

Entrega final: O desenvolvimento iterativo pode ter uma entrega final menos clara após cada iteração, enquanto o desenvolvimento incremental resulta em uma entrega final claramente definida após a conclusão de todos os incrementos planejados.

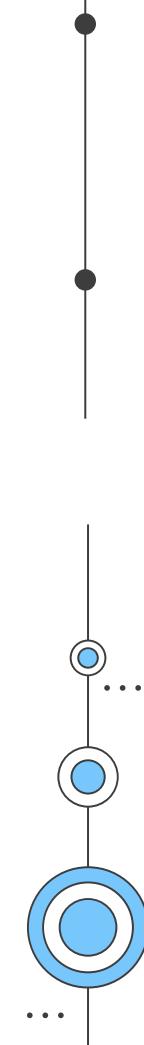




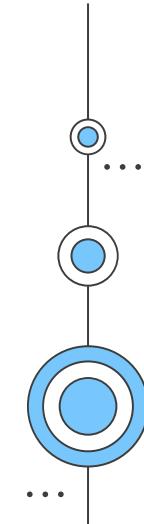
1.7 Metodologia para desenvolvimento e testes

Diferenças:

Prototipagem: O desenvolvimento iterativo pode envolver prototipagem nas primeiras iterações, enquanto o desenvolvimento incremental se concentra na construção sequencial de funcionalidades.



Vejamos um exemplo de protótipo da Buser, utilizando o figma.

- <https://www.figma.com/community/file/1256699010820046769/prototipo-ui-ux-buser>
 - <https://www.buser.com.br/>
 - <https://www.figma.com/>
- ...
- 

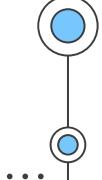
1.7 Metodologia para desenvolvimento e testes

The image displays a grid of 15 screenshots from the Buser app, arranged in three rows. The first row shows the initial login screen, followed by a color selection screen, and then three screens related to travel opportunities: "Encontre milhares de possibilidades para sua próxima viagem", "Receba bonificações por suas viagens e mantenha seu saldo no aplicativo", and "Troque seus pontos acumulados por descontos ou prêmios". The second row shows the registration process: "Cadastrar-se" (with fields for name, email, and password), "Recuperação de senha" (password recovery), and two confirmation screens ("Conta criada com sucesso!" and "Email enviado com sucesso!"). The third row shows the Points section: "Meus pontos" (displaying 5780 points), a promotional offer ("40% OFF: Chácara (SC) - Porto Alegre (RS)"), and a successful gift redemption ("Pedido realizado com sucesso!" with a "Resgatar gift" button).



...

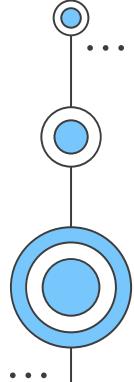
1.7 Metodologia para desenvolvimento e testes



Ambas as abordagens têm suas vantagens e são escolhidas com base nas necessidades específicas do projeto, nos requisitos do cliente e nas características do ambiente de desenvolvimento.

Em muitos casos, equipes adotam práticas que combinam elementos de ambas as abordagens para obter os benefícios da flexibilidade, feedback contínuo e entregas frequentes.

...



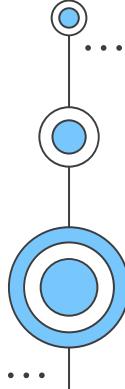


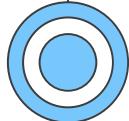
1.7 Metodologia para desenvolvimento e testes

Outras metodologias de desenvolvimento:

Extreme Programming (XP):

- O XP é uma abordagem ágil que enfatiza práticas como desenvolvimento contínuo, programação em pares, testes automatizados e feedback contínuo.





...

1.7 Metodologia para desenvolvimento e testes



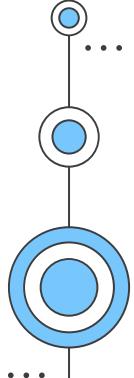
Outras metodologias de desenvolvimento:



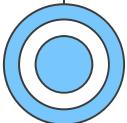
Modelo Cascata (Waterfall):

- O modelo cascata é uma abordagem sequencial, em que cada fase do desenvolvimento é realizada em uma ordem específica.
- Cada fase, como análise, design, implementação, testes e manutenção, só começa quando a fase anterior é concluída.

...



...



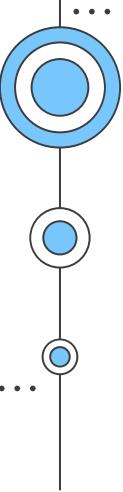
1.7 Metodologia para desenvolvimento e testes

Outras metodologias de desenvolvimento:

Modelo Espiral:

- O modelo espiral combina elementos do modelo cascata com a iteração.
- Ele incorpora ciclos de planejamento, risco, engenharia e avaliação, permitindo ajustes contínuos durante o desenvolvimento.

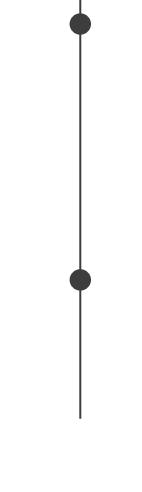
...



1.7 Metodologia para desenvolvimento e testes

Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

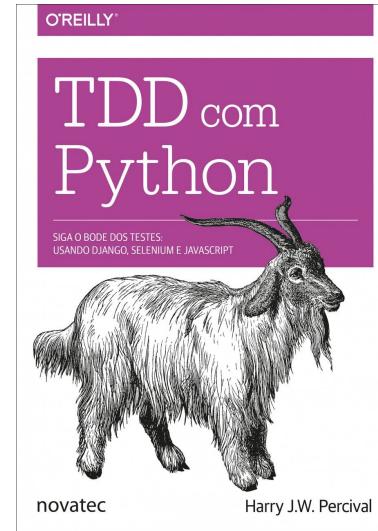
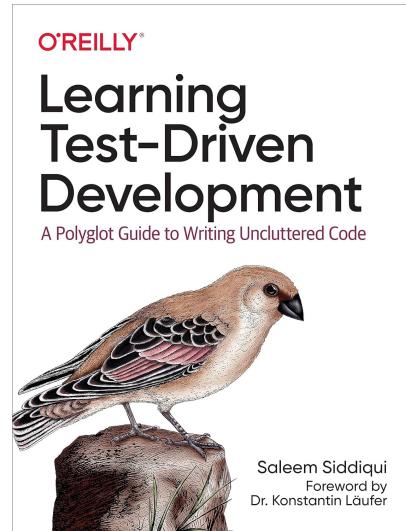
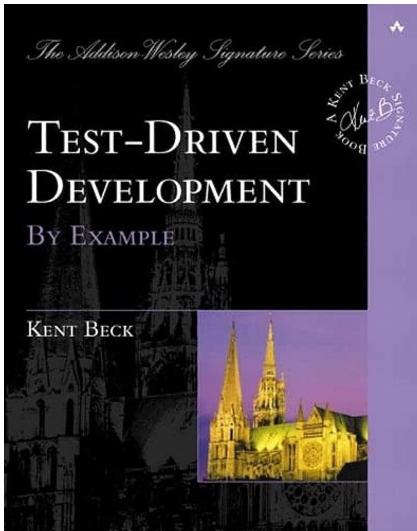
- O Desenvolvimento Orientado a Testes, conhecido como TDD (Test-Driven Development), é uma prática de desenvolvimento de software que coloca a criação de testes automatizados no centro do ciclo de vida do desenvolvimento.
 - A principal ideia por trás do TDD é escrever testes **antes** de implementar o código funcional, criando uma abordagem iterativa e incremental para o desenvolvimento.
- 

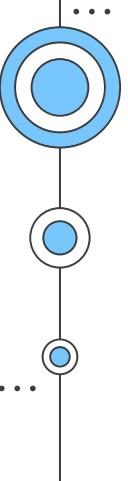
1.7 Metodologia para desenvolvimento e testes

Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- Leituras sugeridas:





1.7 Metodologia para desenvolvimento e testes

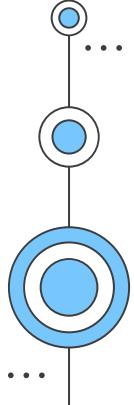


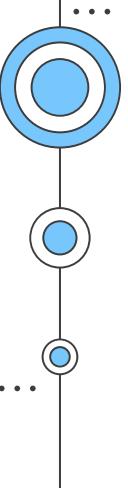
Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- O ciclo típico do TDD é frequentemente resumido nas seguintes etapas:
 - **Red** (Vermelho): Escreva um teste automatizado que falhe inicialmente. Esse teste deve representar uma funcionalidade específica que se deseja implementar.
 - **Green** (Verde): Implemente o código mínimo necessário para fazer o teste passar. O objetivo é tornar o teste verde (ou seja, bem-sucedido).

...

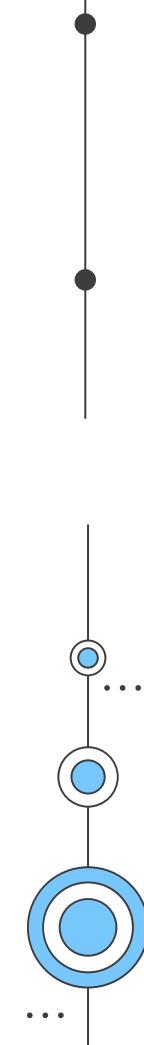


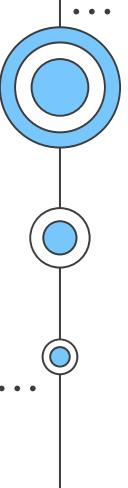


1.7 Metodologia para desenvolvimento e testes

Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- O ciclo típico do TDD é frequentemente resumido nas seguintes etapas:
 - **Refactor** (Refatorar): Refatore o código, se necessário, para melhorar sua estrutura ou eficiência, sem alterar seu comportamento. O código deve permanecer verde após a refatoração.
 - **Repetir**: Repita esse ciclo para cada nova funcionalidade ou alteração no código.
- 



1.7 Metodologia para desenvolvimento e testes

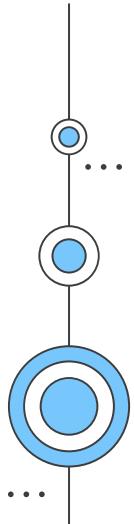


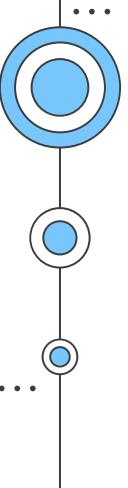
Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- Principais características e benefícios do TDD:
 - **Garantia de qualidade:** Testes automatizados são criados antes da implementação, garantindo que o código seja testado continuamente à medida que é desenvolvido.
 - **Feedback rápido:** O ciclo curto de desenvolvimento fornece feedback rápido sobre a integridade e funcionalidade do código.

...





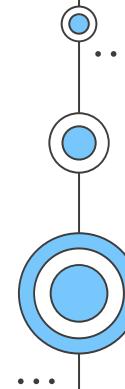
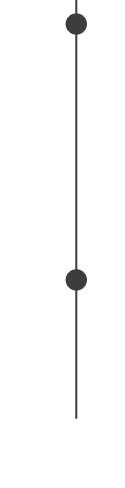
1.7 Metodologia para desenvolvimento e testes

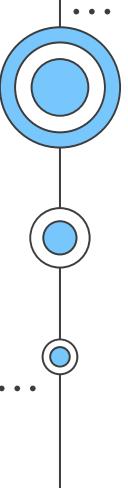
Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- Principais características e benefícios do TDD:
 - **Detecção antecipada de erros:** Erros são identificados e corrigidos rapidamente, uma vez que os testes são executados automaticamente após cada alteração no código.
 - **Melhoria contínua:** A prática de refatoração constante ajuda a melhorar a qualidade do código ao longo do tempo.

...





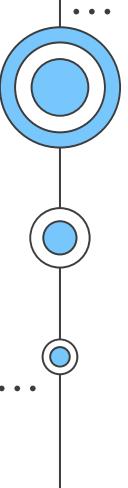
1.7 Metodologia para desenvolvimento e testes



Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- Principais características e benefícios do TDD:
 - **Documentação executável:** Os testes servem como uma forma de documentação executável, descrevendo o comportamento esperado do código.
 - **Adaptação a mudanças:** O TDD facilita a adaptação a mudanças nos requisitos, pois o código é construído em incrementos pequenos e testado continuamente.
- ...

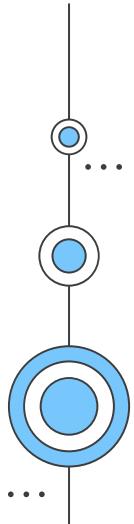


1.7 Metodologia para desenvolvimento e testes



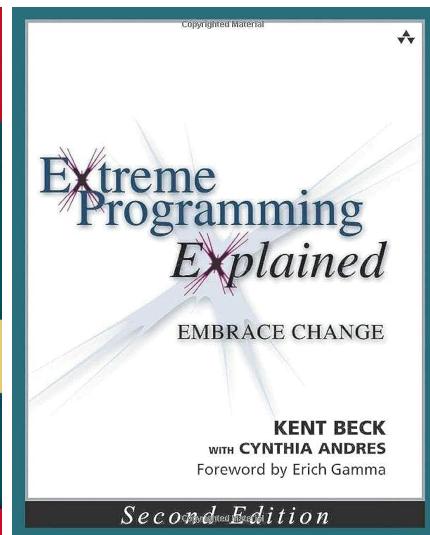
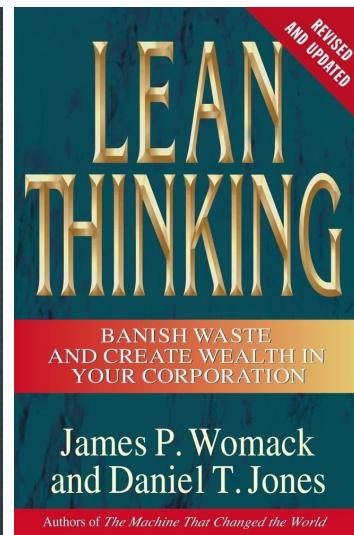
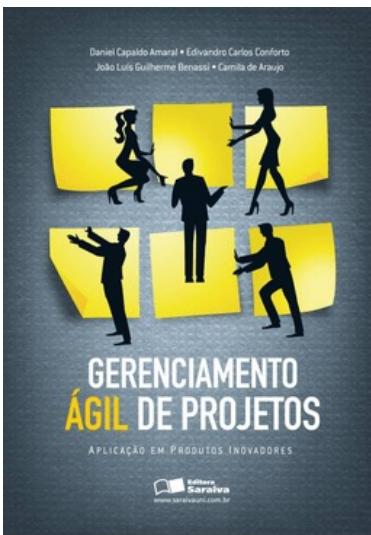
Outras metodologias de desenvolvimento:

Desenvolvimento Orientado a Testes (TDD):

- O TDD é comumente associado a metodologias ágeis, como Scrum e Extreme Programming (XP), mas pode ser aplicado em diferentes contextos e projetos.
 - É uma prática que promove a criação de software mais confiável, testável e adaptável.
- 

1.7 Metodologia para desenvolvimento e testes

Leituras sugeridas:





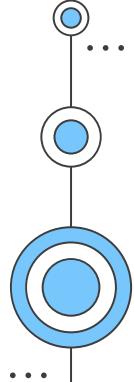
Disciplina – Algoritmos e Estruturas de Dados I

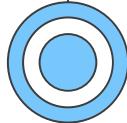


Referências básicas:

- CORMEN, Thomas. **Algoritmos** - Teoria e Prática. Editora GEN LTC. 3^a edição, 2012.
- BHARGAVA, Aditya Y. **Entendendo Algoritmos**: Um Guia Ilustrado Para Programadores e Outros Curiosos. Novatec Editora, 2017.
- BACKES, André Ricardo. **Algoritmos e Estruturas de Dados em Linguagem C**. Editora LTC, 2022.

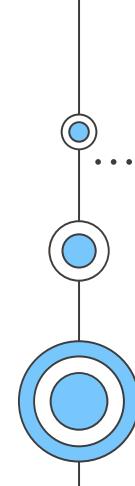
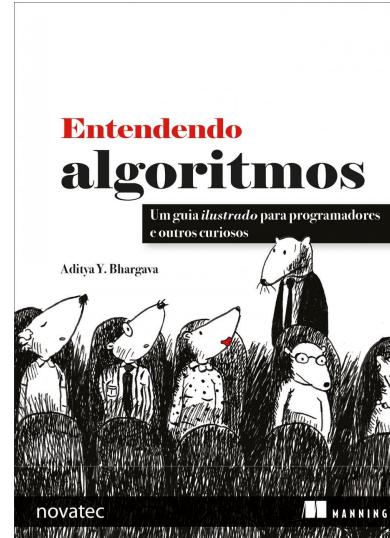
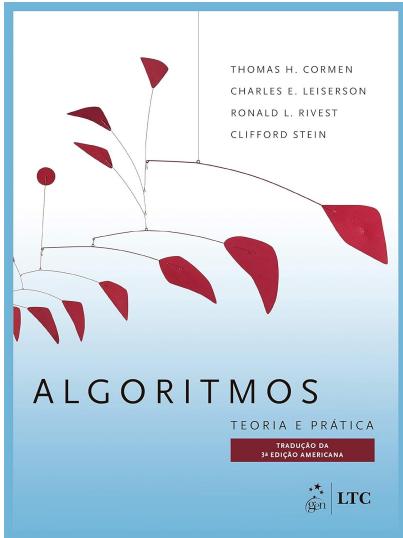
...

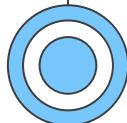




Disciplina – Algoritmos e Estruturas de Dados I

Referências básicas:





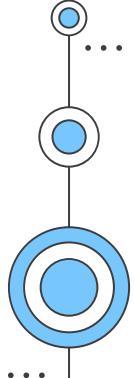
Disciplina – Algoritmos e Estruturas de Dados I

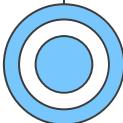


Referências complementares:

- GOODRICH, Michael T. **Estruturas de Dados e Algoritmos em Java**. Editora Bookman. 5^a edição, 2013.
- AGARWAL, Basant. **Estruturas de Dados e Algoritmos com Python**: Armazene, manipule e acesse dados de forma eficaz e melhore o desempenho de suas aplicações. Novatec Editora, 2023.
- GRONER, Loiane. **Estruturas de Dados e Algoritmos com JavaScript**: Escreva um Código JavaScript Complexo e Eficaz Usando a Mais Recente ECMAScript. Novatec Editora, 2019.

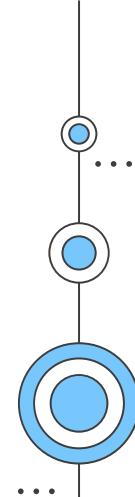
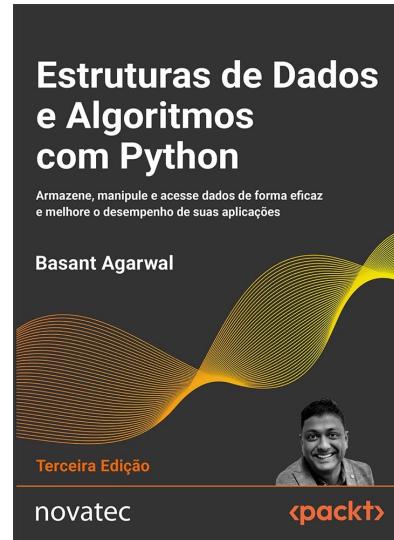
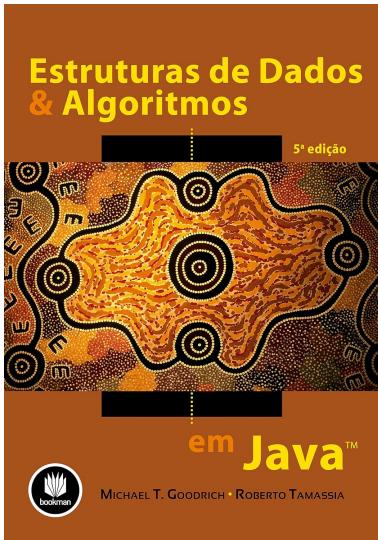
...

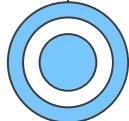




Disciplina – Algoritmos e Estruturas de Dados I

Referências complementares:





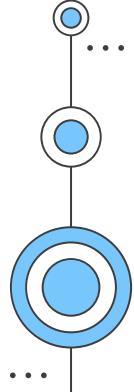
Disciplina – Algoritmos e Estruturas de Dados I



Outras referências:

- PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software**. Editora Grupo AMGH, 2021.
- FOWLER, Martin. **Refatoração**: Aperfeiçoando o Design de Códigos Existentes. Editora Novatec, 2020.
- MARTIN, Robert C. **Código limpo**: habilidades práticas do Agile software. Editora Alta Books, 2009.

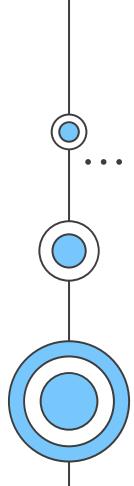
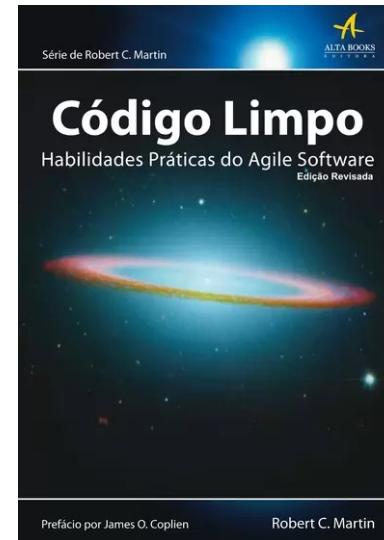
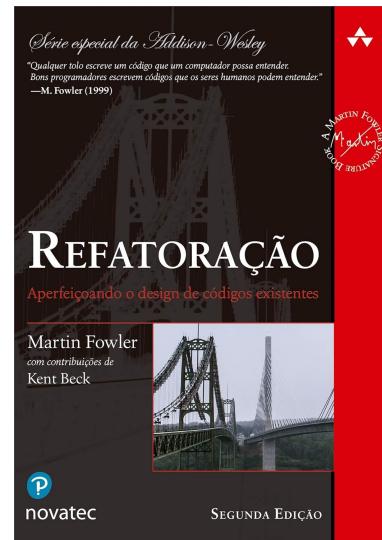
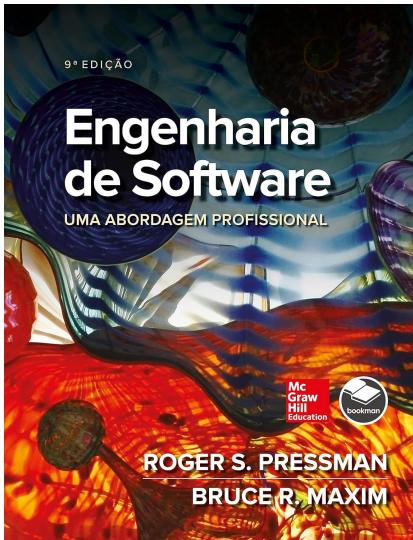
...





Disciplina – Algoritmos e Estruturas de Dados I

Outras referências:





Disciplina – Algoritmos e Estruturas de Dados I

...

...

Nesta disciplina usaremos: Linguagem C (ANSI Style)

- Linguagem C

[DevDocs] <https://devdocs.io/c/>

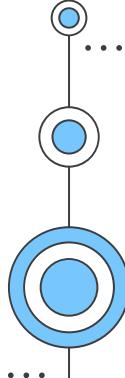
- Eclipse IDE

[Download] <https://www.eclipse.org/downloads/packages/release/2023-12/r/eclipse-ide-cc-developers>

[Docs] <https://www.eclipse.org/documentation/>



...



Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com



[GitHub](#)



[LinkedIn](#)



[Lattes](#)

...



...