

# Programação Aplicada à Estatística

Pedro Rafael Diniz Marinho

Universidade Federal da Paraíba

2021.2

# Sobre mim

Sou Pedro Rafael D. Marinho e serei o professor de vocês nesse período letivo. Sou Dr. em Estatística. Toda minha formação foi na área de Estatística (bacharelado, mestrado e doutorado).

Sou lotado no Centro de Ciências Exatas e da Natureza no Departamento de Estatística - UFPB.

A minha sala é a de número 12.

**Meu email:** [pedro.rafael.marinho@gmail.com](mailto:pedro.rafael.marinho@gmail.com)

# Sobre mim

Trabalhei no mestrado com **modelos lineares com heteroscedasticidade de forma desconhecida** com orientação do PhD Francisco Cribari Neto. Na dissertação desenvolvemos simulações de bootstrap (simples e duplo) para avaliação da cobertura dos intervalos de confiança que indexam os parâmetros desses modelos. Um pacote na linguagem R foi desenvolvido.

## Título da dissertação

Estimadores Intervalares Sob Heteroscedasticidade de Forma Desconhecida Via Bootstrap Duplo

O pacote `hcci` na versão 1.0.0 encontra-se hospedado no CRAN do R em <https://cran.r-project.org/web/packages/hcci/index.htm>.

# Sobre mim

No doutorado trabalhei com o PhD Gauss Moutinho Cordeiro na área de **distribuições de probabilidade**. Na tese foram criados novas classes de distribuições de probabilidade em que é possível gerar uma nova distribuição a partir de uma distribuição  $G$  conhecida (baseline). Também foi construído o pacote AdequacyModel na linguagem R que encontra-se atualmente na versão 2.0.0 sob os termos da licença  $\text{GPL} \geq 2$  (*GNU General Public License*)

## Título da tese

Some New Families of Continuous Distributions

O pacote poderá ser obtido em <https://cran.r-project.org/web/packages/AdequacyModel/index.html>.

# Ementa

O curso de **Programação Aplicada à Estatística** é formado pela seguinte ementa.

## Ementa do Curso

Introdução: Modelo de um computador digital; Linguagem de máquina; Introdução à Programação; Histórico das linguagens de programação; Compiladores e Interpretadores; Lógica e Lógica de Programação; Construção de algoritmos; Pseudocódigo.

Linguagem C: Visão geral; Expressões; Controle de fluxo; Funções; Ponteiros, Vetores e Matrizes; Alocação dinâmica de memória; Cadeias de caracteres; Tipos estruturados; Arquivos; Ordenação e Busca. Aplicações práticas à Estatística.

# Plano de Curso

O **plano de curso** seguirá a ementa apresentada logo acima no *frame* anterior.

O plano de curso é de responsabilidade do professor da disciplina e deve estar de acordo com a ementa da disciplina.

O plano de curso é estabelecido pelo professor e ficará a cargo desse a sua construção e alteração, se necessário, no decorrer do curso, desde que o mesmo esteja de acordo com a ementa do curso.

O rigor e profundidade do assunto ficará a cargo do professor e faz parte do seu plano de curso.

## Plano adotado no momento

1. Modelo de um computador digital.
2. Elementos básicos de um computador típico.
3. Armazenamento de dados e programas na memória.
4. Linguagem de máquina.
5. Histórico das linguagens de programação.
6. Tipos de linguagens.
7. Compiladores e Interpretadores.
8. Lógica e Lógica de programação
9. Construção de algoritmos.
10. Linguagem C.
11. Variáveis.
12. Operadores.
13. Entrada e Saída.
14. Controle de fluxo.
15. Construções de Laços.
16. Seleção.
17. Funções.
18. Ponteiros, Vetores e Matrizes.
19. Alocação dinâmica de memória.
20. Cadeia de caracteres.
21. Tipos estruturados.
22. Arquivos.
23. Ordenação e busca.
24. Aplicações práticas à Estatística.

# Bibliografias Utilizadas

Abaixo estão enumeradas as bibliografias **básicas** utilizadas:

- ① Celes, W., Cerqueira, R., Rangel, J.L. Introdução a Estruturas de Dados: Com Técnicas de Programação em C. Rio de Janeiro: Elsevier, 2004.
- ② Kernighan, B. W., Ritchie, D. M. C: A linguagem de programação padrão ANSI. Rio de Janeiro: Elsevier, 1989.
- ③ Schildt, H. C Completo e Total. São Paulo: Makron Books, 1996.

# Bibliografias Utilizadas

Abaixo estão enumeradas as bibliografias **básicas** utilizadas:

- ① Oliveira, U. Programando em C Volume I: Fundamentos. Rio de Janeiro: Ciência Moderna, 2008.
- ② Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. Numerical recipes in C: the art of scientific computing, Cambridge : Cambridge University Press, 1994.

# Recursos utilizados

Boa parte do curso será apoiada pelo uso de **datashow** o que nos ajudará bastante a decorrer sobre os diversos assuntos contidos na ementa desse curso que é bastante ampla.

O quadro será utilizado para resolução de alguns exemplos bem como complementações em que o professor achar conveniente no momento de aula.

# Sobre as avaliações

No curso iremos considerar **três avaliações**, duas provas e **TALVEZ** um trabalho.

As duas primeiras avaliações serão provas que irão contemplar o conteúdo ministrado em sala de aula (**tudo que foi dito, apresentado e escrito**).

No caso em que for decidido por um trabalho como terceira avaliação, este deverá ser formado por um grupo de no máximo 3 pessoas.

# Sobre as avaliações

As datas e temas dos trabalhos serão fornecidos após a segunda avaliação. A divisão dos grupos ficará a cargo dos alunos.

**O professor tomará partido na divisão em caso de problemas.**

Os trabalhos serão avaliados segundo sua organização, profundidade do assunto e irei avaliar o código que será fornecido no Apêndice do trabalho e enviado para meu email.

## Importante

A reposição do trabalho será uma prova referente ao assunto do tema da equipe em que o aluno encontra-se presente.

# Sobre as reposições

O aluno terá direito a apenas uma reposição de uma das avaliações desde que satisfeito o que rege a **Resolução N° 16/2015 que aprova o Regulamento dos Cursos Regulares de Graduação da UFPB.**

O aluno poderá repor uma prova desde que entre com pedido de reposição junto à coordenação do seu curso. O coordenador de seu curso irá avaliar o pedido de reposição com base na **Resolução N° 16/2015** do CONSEPE e encaminhará o pedido julgado ao Departamento de Estatística - UFPB.

Apenas irá repor a prova quem atender os requisitos do Art. 92, 6º §.

# Aos alunos interessados ( $\text{\LaTeX}$ )

Aconselho fortemente ao aluno que pretende produzir textos de qualidade (qualidade tipográfica) considerar o uso da linguagem de comandos macros  $\text{\LaTeX}$ .

$\text{\LaTeX}$  é uma linguagem de comandos macros de  $\text{\TeX}$  e atualmente encontra-se na versão  $\text{\LaTeX} 2\epsilon$ .

Com o uso de  $\text{\LaTeX}$  a facilidade de construir texto de alta qualidade tipográfica será uma ferramenta a mais na mão de um profissional em estatística.

Com  $\text{\LaTeX}$  é possível fazer grandes mudanças em um texto em poucos minutos apenas acrescentando alguns comandos ao preâmbulo do código.

# Aos alunos interessados ( $\text{\LaTeX}$ )



Figura: Donald Knuth.

$\text{\TeX}$  é um sistema de tipografia científica desenvolvido por **Donald E. Knuth** que é orientado à produção de textos técnicos e fórmulas matemáticas. A pedido da AMS (*American Mathematical Society*), Donald Knuth desenvolveu uma linguagem de computador para editoração de textos com muitas equações.

# Aos alunos interessados ( $\text{\LaTeX}$ )



Figura: Donald Knuth.

O trabalho de criação do  $\text{\TeX}$  se estendeu de 1977 a 1998, quando  $\text{\TeX}$  foi disponibilizado gratuitamente. O  $\text{\TeX}$  possui aproximadamente **600 comandos** que controlam a construção de uma página.

Pode-se considerar o  $\text{\TeX}$  como sendo um compilador para textos científicos que produz documentos de alta qualidade tipográfica.

# Aos alunos interessados ( $\text{\LaTeX}$ )

O  $\text{\TeX}$  atingiu um estado de desenvolvimento em que Beebe (1990 afirmou):

“Meu trabalho no desenvolvimento de  $\text{\TeX}$ , METAFONT e as fontes Computer Modern chegou ao final. Eu não irei realizar mudanças futuras, exceto corrigir sérios erros de programação.”

**Ver em:** BEEBE, N. H. Comments on the future of TeX and METAFONT. TUGboat, v. 11, n. 4, p. 490–494, 1990.

# Aos alunos interessados ( $\text{\LaTeX}$ )

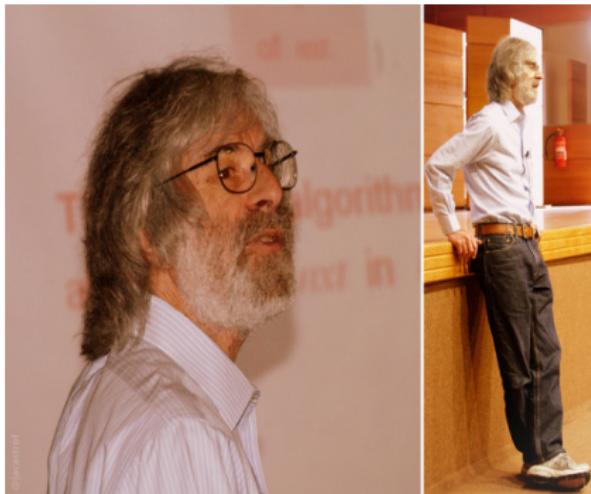


Figura: Leslie Lamport.

Quase que em paralelo foi desenvolvido por Leslie Lamport o  $\text{\LaTeX}$ . Essas macros definem tipos de documentos, tais como livros, artigos, cartas, entre outros.

Inclusive essa apresentação é um tipo básico de documento que foi produzido em  $\text{\LaTeX}$ .

# Aos alunos interessados ( $\text{\LaTeX}$ )

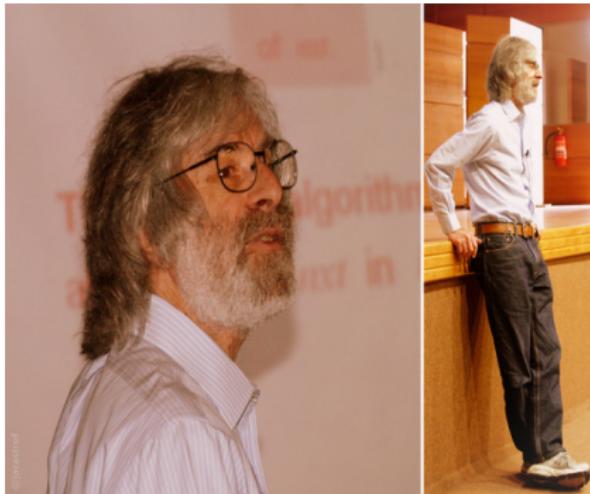


Figura: Leslie Lamport.

Para maiores detalhes leia sobre o pacote `beamer` que está disponível com a maioria das distribuições  $\text{\LaTeX}$ . Há diversos documentos disponíveis nos mais variados idiomas na rede.

`beamer` também está disponível no **The Comprehensive  $\text{\TeX}$  Archive Network (CTAN)**.

# Aos alunos interessados ( $\text{\LaTeX}$ )



Figura: CTAN lion drawing  
by Duane Bibby.

**CTAN** é o lugar central para todos os tipos de material em torno de  $\text{\TeX}$ . CTAN tem atualmente **5752 pacotes** e **2637 colaboradores** contribuíram para essa quantidade de pacotes.

O símbolo ao lado foi desenhado pelo artista comercial Duane Bibby. Este leão foi utilizado nas ilustrações para o livro  $\text{\TeX}Book$  de Donald Knuth e apareceu com grande frequência em outros materiais.

# Aos alunos interessados ( $\text{\LaTeX}$ )



Figura: CTAN lion drawing  
by Duane Bibby.

Maiores detalhes sobre o CTAN podem ser encontrados em <https://www.ctan.org/lion/>. Desde dezembro de 1994, o pacote  $\text{\LaTeX}$  está sendo atualizado pela equipe  $\text{\LaTeX} 3$ , dirigida por Frank Mittelbach, para incluir algumas melhorias que já vinham solicitadas a algum tempo. A equipe se preocupa também em reunificar todas as versões modificadas que surgiram desde o aparecimento do  $\text{\LaTeX} 2.09$ .

# Aos alunos interessados ( $\text{\LaTeX}$ )



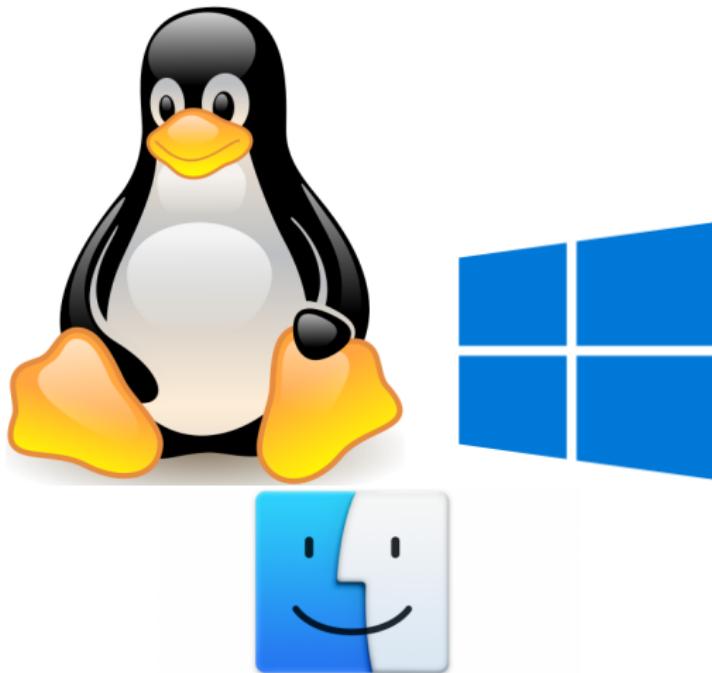
Figura: CTAN lion drawing  
by Duane Bibby.

O melhor de tudo, o  $\text{\LaTeX}$  é um sistema estável mas com crescimento constante, podendo ser instalado em quase todos os sistemas operacionais.

Usuários de Unix, Linux, Windows ou Mac OS X podem dispor de todo ferramental para produzir ótimos textos com o  $\text{\LaTeX}$ .

**Nota:** Pronuncia-se “leitec” e não “latéx”.

Aos alunos interessados ( $\text{\LaTeX}$ )



Mac<sup>TM</sup> OS

# Como instalo o L<sup>A</sup>T<sub>E</sub>X no Linux?

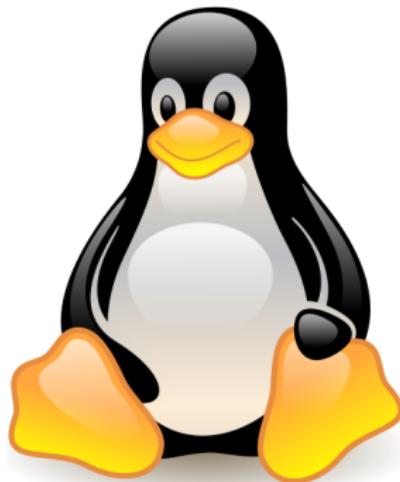


Figura: Tux (Mascote do Linux).

# Como instalo o L<sup>A</sup>T<sub>E</sub>X no Linux?

Inicialmente é preciso instalar o compilador de L<sup>A</sup>T<sub>E</sub>X. Recomendo o uso do T<sub>E</sub>X Live.

A maioria das distribuições linux (Arch, Ubuntu, Fedora, Mint, Sabayon, entre outros “sabores”) apresentam esse compilador de L<sup>A</sup>T<sub>E</sub>X em seus repositórios.

Por exemplo, no **Arch Linux** e distribuições derivadas que utilizam os mesmos repositórios do Arch como Antergos façam:

```
sudo pacman -S texlive.
```

# Como instalo o L<sup>A</sup>T<sub>E</sub>X no Linux?

No **Ubuntu** ou qualquer distribuição que faz uso dos repositórios do Ubuntu façam:

```
sudo apt-get install texlive-full.
```

Já os usuários da distribuição **Fedora** e distribuições derivadas que utilizam-se dos mesmos repositórios devem fazer:

```
sudo dnf -y texlive-scheme-full.
```

**Observação:** Todos os comandos acima devem ser executados no terminal da respectiva distribuição com permissão de super usuário (usuário que pode fazer alterações no sistema operacional).

# Como instalo o L<sup>A</sup>T<sub>E</sub>X no Windows?



# Como instalo o L<sup>A</sup>T<sub>E</sub>X no Windows?

Felizmente, há o T<sub>E</sub>XLive para Windows que poderá ser obtido no site oficial do projeto T<sub>E</sub>XLive.

O usuário de Windows deverá baixar o arquivo

`install-tl-windows.exe`

que possui aproximadamente **13mb**.

**Nota:** `install-tl-windows.exe` é apenas o instalador do T<sub>E</sub>XLive para Windows. Dessa forma, ao final da instalação, o T<sub>E</sub>XLive terá muito mais que apenas 13mb instalado em seu computador.

# Como instalo o L<sup>A</sup>T<sub>E</sub>X no Linux?

Mas para escrevermos um texto com qualidade usando o L<sup>A</sup>T<sub>E</sub>X precisamos também de um editor de texto.

Na maioria dos casos usamos uma **IDE** (*Integrated Development Environment*) (**Ambiente de Desenvolvimento Integrado**)

Aconselho o uso do T<sub>E</sub>Xstudio que está disponível para Linux, Windows e Mac OS.

O T<sub>E</sub>Xstudio é um software sobre os termos da licença **GPL** (*GNU General Public License*) e pode ser obtido em  
<http://texstudio.sourceforge.net/>.

# TEXstudio

**Observação:** Aperte F6 para compilar o documento e F7 para visualizar o PDF produzido por meio de código LATEX.

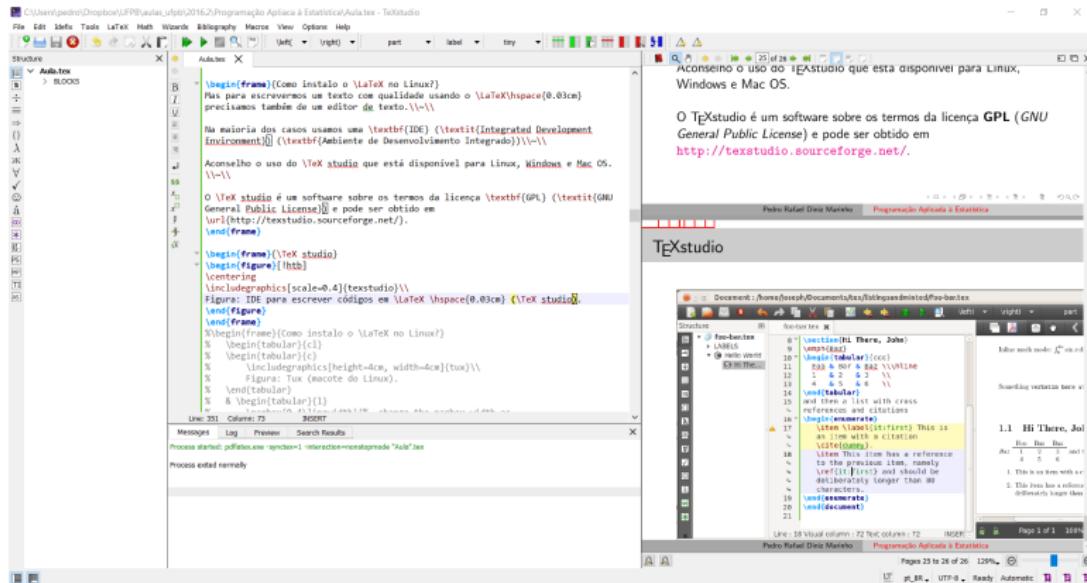


Figura: IDE para escrever códigos em LATEX (TeXstudio).

# Vantagens do TEXstudio

# Vantagens do TEXstudio

- ① É uma IDE leve.

# Vantagens do TEXstudio

- ① É uma IDE leve.
- ② O TEXstudio auto-completa os comandos que vão sendo digitados.

# Vantagens do TEXstudio

- ① É uma IDE leve.
- ② O TEXstudio auto-completa os comandos que vão sendo digitados.
- ③ É possível ir de um ponto específico do código LATEX para o ponto correspondente no PDF criado.

# Vantagens do TEXstudio

- ① É uma IDE leve.
- ② O TEXstudio auto-completa os comandos que vão sendo digitados.
- ③ É possível ir de um ponto específico do código LATEX para o ponto correspondente no PDF criado.
- ④ Tem licença **GPL** (*GNU General Public License*), isto é, não pago por ela e tenho acesso ao seu código fonte.

# Vantagens do TEXstudio

- ① É uma IDE leve.
- ② O TEXstudio auto-completa os comandos que vão sendo digitados.
- ③ É possível ir de um ponto específico do código LATEX para o ponto correspondente no PDF criado.
- ④ Tem licença **GPL** (*GNU General Public License*), isto é, não pago por ela e tenho acesso ao seu código fonte.
- ⑤ O PDF é visualizado ao lado do código LATEX.

# Como aprender L<sup>A</sup>T<sub>E</sub>X?

## Pergunta do aluno

Okay professor, o senhor me convenceu em utilizar L<sup>A</sup>T<sub>E</sub>X. Então, como eu posso aprender os seus comandos? Há algum material interessante para se começar a estudar L<sup>A</sup>T<sub>E</sub>X?

## Resposta do professor legal

Há vários livros e materiais disponibilizados na internet. Há diversos grupos de discussão sobre L<sup>A</sup>T<sub>E</sub>X. Para não complicar muito, aconselho estudar por essa apostila:

[http://www.univasf.edu.br/~joseamerico.moura/index\\_arquivos/lenimar\\_tex\\_LATEX.pdf](http://www.univasf.edu.br/~joseamerico.moura/index_arquivos/lenimar_tex_LATEX.pdf)

# Programação

## O que é programação?

**Resposta:** Linguagem de programação é um método padronizado para comunicar instruções para um computador por meio de uma sintaxe.

Trata-se de um conjunto de regras sintáticas utilizadas para passar instruções para um computador. Por meio dessas regras, é possível que o programador especifique os **tipos de dados** em que o professor irá processar.

Tais dados serão armazenados e transmitidos e/ou transmitidos entre os componentes que formam o computador. Assim, a linguagem também permite especificar quais ações devem ser tomadas e sob quais circunstâncias serão tomadas.

# Linguagens



Figura: Diversas linguagens de programação disponíveis para uso.

Por que aprender à programar?

**Por que um estatístico deve saber programar?**

# Por que aprender à programar?

## Por que o estatístico deve saber programar?

**Resposta:** Simplesmente pelo fato de que **não** dá para fazer muita coisa na estatística (principalmente no mercado de trabalho) se o profissional não é capaz de fazer com que o computador resolva os seus problemas.

## Um aluno questiona...

Mas professor, temos o SPSS, Excel, SAS, Statistica e outros softwares em que podemos chamar nosso conjunto de dados e apertar centenas de botões e ter **alguns** resultados.

# Por que aprender à programar?

## O professor continua respondendo...

A maioria desses software não possuem as técnicas estatísticas mais recentes e muitas vezes não são capazes de se adequar aos problemas específicos que nos deparamos ao tentar resolver um problema.

Muitas vezes precisamos modificar uma função programada por uma outro programador para que ela venha a funcionar no nosso problema.

**Observação:** Diversas outras vezes precisamos programar para realizar simulações. É muito comum na estatística estudar propriedades de algumas estatística ou modelo estatístico e querer simular o seu comportamento em diversos cenários diferentes.

# Por que aprender à programar?

Uma linguagem de programação bastante utilizada na estatística é a linguagem R.

## Linguagem R

R é uma linguagem de programação para computação estatística e gráficos. R é uma parte oficial do projeto GNU da Free Software Foundation's.

**Curiosidade:** A linguagem R foi criada originalmente por Ross Ihaka e Robert Gentleman no **Departamento de Estatística** da Universidade de Auckland, Nova Zelândia em agosto de 1993.

**Nota:** É muito importante que um estatístico saiba programar na linguagem R. Alguns empregos exigem isso. Porém, se não exigirem, o R te ajudará bastante.

# Por que aprender à programar?



Figura: Criadores da linguagem R [Robert Gentleman (foto à esquerda) e Ross Ihaka (foto à direita)].

# Por que aprender à programar?



Figura: Logo da linguagem R.

Um dos grandes motivos da grande popularidade da linguagem R se deve a grande quantidade de pacotes disponíveis para os usuários da linguagem.

Atualmente há mais de 5 mil pacotes para R com o foco nas mais variadas áreas: estatística, matemática, biologia, economia, entre outras.

# Por que aprender à programar?



Figura: Logo da linguagem R.  
Obtenha a linguagem R em  
<https://www.r-project.org/>.

**Observação:** Para programar em R **não é suficiente entender alguns pacotes específicos**. É preciso entender a sintaxe base da linguagem que nos permite inclusive criar outros pacotes e melhorar os existentes.

# Por que aprender à programar?



Figura: Logo da linguagem R.  
Obtenha a linguagem R em  
<https://www.r-project.org/>.

**Observação:** As novas metodologias estatísticas chegam mais rapidamente em R do que em outros softwares estatísticos pelo fato do R ser uma linguagem livre (código aberto e gratuita).

# Esclarecedor

“Ciência  
da computação tem tanto  
a ver com o computador como  
a Astronomia com o telescópio,  
a Biologia com o microscópio,  
ou a Química com os  
tubos de ensaio. A Ciência não  
estuda ferramentas, mas o que  
fazemos e o que descobrimos  
com elas.” - **Edsger Dijkstra**  
(Prêmio Turing em 1972)



Figure: Edsger Dijkstra

# Máquina de Turing

A máquina de Turing é um dispositivo teórico conhecido como máquina universal, que foi concebido pelo matemático britânico Alan Turing (1912-1954), muitos anos antes de existirem os modernos computadores digitais.



Figure: Exemplo de uma Turing física.

# Máquina de Turing

Na Teoria da Computabilidade, um problema é solúvel se há uma Máquina de Turing para aquele problema

Em seu artigo original, Turing demonstra a existência de um **problema insolúvel**.

Existe basicamente dois tipos de máquinas de Turing:

- ① Máquina de Turing **Determinística**: (Se 'A', então 'B')
- ② Máquina de Turing **Não-Determinística**: (Se 'A', então 'B' **ou** 'C' **ou** 'D' **ou** ... )

# Teoria da Computabilidade

A computabilidade é a Teoria da Complexidade Computacional que estudam os **limites** da computação:

- ① Quais problemas jamais poderão ser resolvidos por um computador, independente de sua velocidade ou memória?
- ② Quais problemas podem ser resolvidos por um computador, mas requerem um período tão extenso de tempo para completar a ponto de tornar a solução impraticável?
- ③ Em que situações podem ser mais difícil resolver um problema do que verificar cada uma das soluções?

# Teoria da Computabilidade

Das três perguntas anteriores, a última é referente às **classes de paradigmas**, são elas:

- Classe **P**: De tempo polinomial determinístico. Os algoritmos pertencentes à esta classe são chamados de **algoritmos eficientes**.
- Classe **NP**: De tempo polinomial não-determinístico.

O conjunto de problemas que não podem ter solução em tempo polinomial mas candidatos a solução podem ser checados em tempo polinomial são problemas pertencentes à classe NP.

Somente uma máquina de Turing não-determinística podem resolver esses problemas. Eles são resolvidos em tempo polinomial por uma máquina de Turing não-determinística que acerta em todos os passos.

# Teoria da Computabilidade

Na matemática, a questão a respeito de  $P = NP$  ou  $P \neq NP$  é um problema em aberto.

A grande importância dessa classe ( $NP$ ) de problemas se baseia no fato de que ela contém muitos problemas de busca e otimização para os quais gostaríamos de saber se há uma solução.

**Exemplos:** Problema do Caixeiro Viajante, problema da mochila, entre outros.

# E se $P = NP$ ?



- ① Diga adeus a criptografia;
- ② Soluções matemáticas não complicadas;
- ③ Previsão do tempo, terremotos e tsunamis;

# Problemas NP (precisa-se de linguagens eficientes)

## Problemas NP-Completos fazem parte de nossas vidas...

Na estatística sempre nos deparamos com problemas NP-Completos, isto é, sempre lidamos com problemas que não possuem soluções em tempo polinomial.

**Exemplo:** Constantemente precisamos estimar parâmetros de um modelo probabilístico por meio do método de máxima verossimilhança, isto é, maximizamos a função de log-verossimilhança de um modelo probabilístico. A otimização global é um problema NP-Completo.

**Observação:** Atualmente está cada vez mais complicados realizar tais otimizações uma vez que os modelos cada vez mais estão adicionando parâmetros extras o que torna a função extremamente complicada em alguns casos.

# Breve História da Linguagem C

Embora possua um nome estranho quando comparada com outras linguagens de programação da terceira geração, como FORTRAN, PASCAL, ou COBOL, a linguagem C é uma das linguagens mais importantes até hoje criada.

**Curiosidade:** O nome da linguagem (e a própria linguagem) resulta da evolução de uma outra linguagem de programação, desenvolvida pelo programador **Ken Thompson** nos Laboratórios Bell, chamada de B.

# Breve História da Linguagem C



Figura: Ken Thompson (sentado) jogando xadrez com um colega.

B trata-se de uma simplificação da linguagem BCPL (*Basic Combined Programming Language*). Assim como BCPL, B só possuía um tipo de dados.

A linguagem B também foi recebida contribuições do Dennis Ritchie (criador de C).

# Breve História da Linguagem C



Figura: Dennis Ritchie (criador da linguagem C).

A linguagem C foi criada em 1972 nos *Bell Telephone Laboratories* por Dennis Ritchie com a finalidade de permitir a escrita do sistema operacional Unix.

Desejava-se uma linguagem de alto nível de modo a evitar o uso do Assembly.

# Breve História da Linguagem C

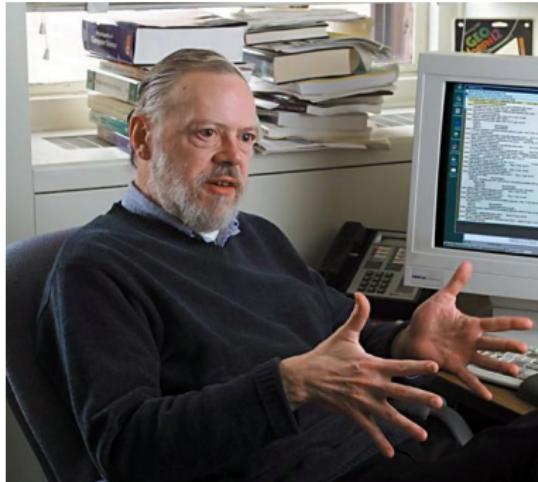


Figura: Dennis Ritchie (criador da linguagem C).

Devido às capacidades e através da divulgação do sistema Unix pelas universidades dos Estados Unidos, a linguagem C deixou cedo as portas dos laboratórios Bell.

C disseminou-se e tornou-se conhecida por todos os tipos de programadores, independentemente dos projetos em que estivessem envolvidos.

# Breve História da Linguagem C

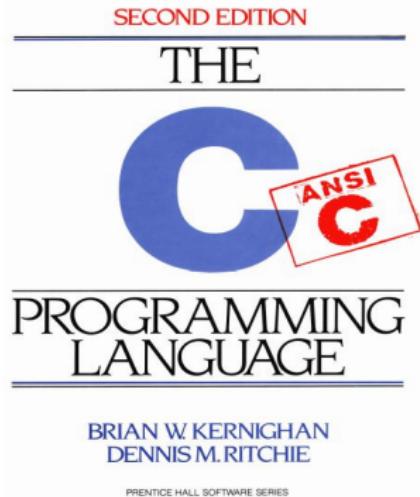


Figura: Ótimo livro sobre a linguagem C.

Essa dispersão de diferentes projetos utilizando a linguagem C levou a que diferentes organizações desenvolvessem e utilizassem diferentes versões da linguagem C criando assim alguns problemas de compatibilidade, entre diversos outros.

O material ao lado é um livro sobre a linguagem C escrito pelo seu criador.

# Breve História da Linguagem C

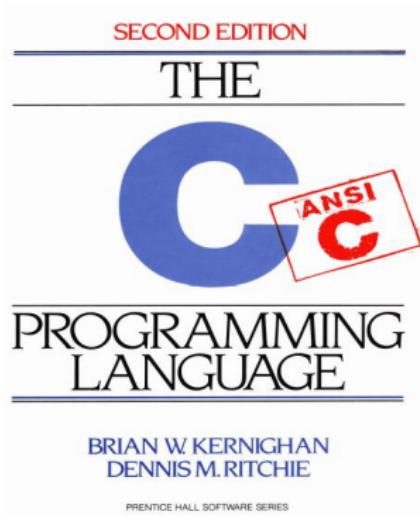


Figura: Ótimo livro sobre a linguagem C.

Devido ao fenômeno que foi a linguagem C e aos problemas de compatibilidade que existiam na época, o **American National Standards Institute** (ANSI) formou em 1983 um comitê para a definição de um padrão para a linguagem C.

# Breve História da Linguagem C

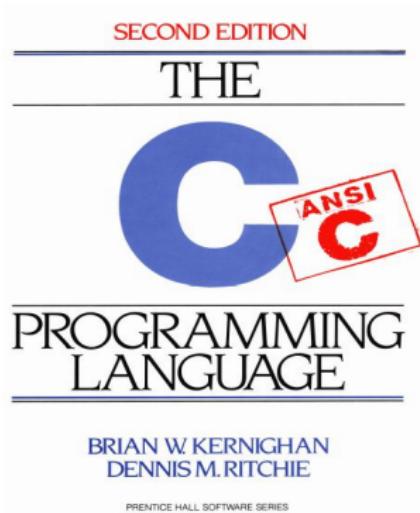


Figura: Ótimo livro sobre a linguagem C.

O padrão tem como objetivo o funcionamento semelhante de todos os compiladores da linguagem, com especificações muito precisas sobre aquilo que a linguagem deve ou não fazer, seus limites, definições, dentre outras coisas.

# Mil e Uma Razões para Programar em C

Devido à enorme quantidade de linguagens de programação disponíveis no mercado, seria necessário que uma delas se destacasse muito em relação às outras para conseguir interessar tantos programadores.

A maior parte das linguagens de programação tem um objetivo específico a atingir:

- PASCAL - Ensino de Técnicas de Programação.
- FORTRAN - Cálculo Científico.
- LISP e PROLOG - Vocacionadas para as áreas de Inteligência Artificial.

# Mil e Uma Razões para Programar em C

## Pergunta do aluno

Certo, entendi, temos linguagens que se destinam a um objetivo específico como as que foram listadas a cima. Mas quanto à C, a que área de desenvolvimento se destina?

# Mil e Uma Razões para Programar em C

## Pergunta do aluno

Certo, entendi, temos linguagens que se destinam a um objetivo específico como as que foram listadas a cima. Mas quanto à C, a que área de desenvolvimento se destina?

## Resposta do professor legal

**NENHUMA** 😊 .

# Por que C?

## Qual o porquê de se aprender a linguagem C

C é uma linguagem bastante interessante devido ser uma linguagem de programação de propósito geral, isto é, C pode ser utilizada para os mais variados fins.

**Alguns outros motivos para se aprender C são:**

# Por que C?

## Qual o porquê de se aprender a linguagem C

C é uma linguagem bastante interessante devido ser uma linguagem de programação de propósito geral, isto é, C pode ser utilizada para os mais variados fins.

## Alguns outros motivos para se aprender C são:

- 1 C é uma linguagem extremamente eficiente.

# Por que C?

## Qual o porquê de se aprender a linguagem C

C é uma linguagem bastante interessante devido ser uma linguagem de programação de propósito geral, isto é, C pode ser utilizada para os mais variados fins.

## Alguns outros motivos para se aprender C são:

- ① C é uma linguagem extremamente eficiente.
- ② Diversas outras linguagem foram programadas utilizando C.  
Por exemplo a linguagem Python, R, entre outras.

# Por que C?

## Qual o porquê de se aprender a linguagem C

C é uma linguagem bastante interessante devido ser uma linguagem de programação de propósito geral, isto é, C pode ser utilizada para os mais variados fins.

### Alguns outros motivos para se aprender C são:

- ① C é uma linguagem extremamente eficiente.
- ② Diversas outras linguagem foram programadas utilizando C.  
Por exemplo a linguagem Python, R, entre outras.
- ③ Na estatística muitas vezes precisamos fazer simulações que são computacionalmente intensivas.

# Por que C?

## Qual o porquê de se aprender a linguagem C

C é uma linguagem bastante interessante devido ser uma linguagem de programação de propósito geral, isto é, C pode ser utilizada para os mais variados fins.

### Alguns outros motivos para se aprender C são:

- ① C é uma linguagem extremamente eficiente.
- ② Diversas outras linguagem foram programadas utilizando C. Por exemplo a linguagem Python, R, entre outras.
- ③ Na estatística muitas vezes precisamos fazer simulações que são computacionalmente intensivas.
- ④ C é uma ótima linguagem para se começar a programar.

# Por que C?

## Qual o porquê de se aprender a linguagem C

C é uma linguagem bastante interessante devido ser uma linguagem de programação de propósito geral, isto é, C pode ser utilizada para os mais variados fins.

### Alguns outros motivos para se aprender C são:

- ① C é uma linguagem extremamente eficiente.
- ② Diversas outras linguagem foram programadas utilizando C. Por exemplo a linguagem Python, R, entre outras.
- ③ Na estatística muitas vezes precisamos fazer simulações que são computacionalmente intensivas.
- ④ C é uma ótima linguagem para se começar a programar.
- ⑤ Códigos em C podem ser importados para R, isto é, R “conversa” com C.

# Por que C?

**Observação:** Pelo fato de C ser uma linguagem de propósito geral (*general purpose*), esta linguagem pode ser utilizada nos mais variados fins, como sistemas operacionais, interfaces gráficas, etc.

## Importante

Há uma falácia de que C é uma linguagem extremamente difícil. Na verdade ocorre que muitas pessoas começam a estudar programação por meio de C, momento este em que é somado dificuldades em aprender à programar (lógica de programação) com as dificuldades de se aprender uma sintaxe de uma linguagem de programação.

**Observação:** Porém, é verdade que muitas coisas temos que fazer nós mesmo em C. Por isso que ela é a linguagem adotada na maioria dos cursos de introdução à programação espalhados pelo mundo.

# Por que C?

## Outras razões para se utilizar C

C é utilizado quando a velocidade, espaço e portabilidade são importantes. A maioria dos sistemas operacionais das outras linguagens e de grande parte dos softwares e games são escritas em C.

**Observação:** Há basicamente três padrões de C que podem ser encontrados por aí. São eles:

- ① **ANSI C** que é do fim dos anos de 1980 e é utilizado para códigos mais antigos;
- ② Muitas coisas foram consertadas no padrão C99 de 1999;
- ③ Algumas novidades foram acrescentadas no atual padrão C11 lançado em 2011.

**Observação:** Não existem grandes diferenças entre as versões de C. Iremos destacá-las ao longo do caminho.

# Arquitetura de von Neumann



Figura: John von Neumann.

A arquitetura de von Neumann é uma arquitetura de um computador digital que possibilita uma máquina digital armazenar os seus programas no mesmo espaço de memória que os dados, podendo assim manipular tais programas.

“Não há sentido em ser preciso quando não se sabe sobre o que está a falar” - von Neumann.

# Arquitetura de von Neumann

A máquina proposta por von Neumann possui as seguintes componentes:

- ① Uma **memória**;
- ② Uma **unidade aritmética e lógica**;
- ③ Uma **unidade central de processamento** (CPU), composto por diversos registradores;
- ④ Uma **unidade de controle**, cuja função é a mesma da tabela de controle de uma Máquina de Turing universal (estabelece as mudanças de estado por meio das entradas).

# Compilador

## O que é um compilador?

# Compilador

## O que é um compilador?

**Resposta:** Um compilador é um programa de computador ou mesmo um grupo de programas que é responsável por traduzir um código fonte escrito em uma linguagem compilada à um programa equivalente do ponto de vista semântico.

# Compilador

## O que é um compilador?

**Resposta:** Um compilador é um programa de computador ou mesmo um grupo de programas que é responsável por traduzir um código fonte escrito em uma linguagem compilada à um programa equivalente do ponto de vista semântico.

O compilador traduz o código fonte de uma linguagem de programação de médio/alto nível para uma linguagem de programação de baixo nível (a exemplo da linguagem Assembly ou código de máquina).

# Compilador

Bytecode?

## Bytecode?

Alguns compiladores traduzem o código para um formato intermediário, denominado de **bytecode** que é um código de baixo nível. Sendo assim, o bytecode não é imediatamente um arquivo executável.

## Bytecode?

Alguns compiladores traduzem o código para um formato intermediário, denominado de **bytecode** que é um código de baixo nível. Sendo assim, o bytecode não é imediatamente um arquivo executável.

**Observação:** Chamamos de linguagem de baixo nível as linguagens que trabalhamo próximo ao hardware. Baixo nível, médio nível ou alto nível em nada tem a ver com a qualidade da linguagem de programação.

## Bytecode?

Alguns compiladores traduzem o código para um formato intermediário, denominado de **bytecode** que é um código de baixo nível. Sendo assim, o bytecode não é imediatamente um arquivo executável.

**Observação:** Chamamos de linguagem de baixo nível as linguagens que trabalhamo próximo ao hardware. Baixo nível, médio nível ou alto nível em nada tem a ver com a qualidade da linguagem de programação.

**Importante:** Jamais confunda bytecode com código de máquina. Bytecode é um formato intermediário que irá ser interpretado em uma máquina virtual que fará a execução.

# Compilador

A vantagem do bytecode é que o código torna-se mais **portável**, isto é, podemos com o resultado da compilação executar o código proveniente de um processo de compilação em diversas arquiteturas distintas. Dessa forma, o bytecode irá produzir o mesmo resultado esperado em qualquer arquitetura que possua uma máquina virtual que execute o código intermediário.

**Exemplos de linguagem que converte o código fonte para bytecode:** Java que corre o código sobre a máquina virtual Java, .NET que corre o código sobre a *Common Language Runtime*.

## Código Objeto e Código de Máquina

O **código de máquina** é um código binário (0 e 1) que poderá ser executado diretamente pela CPU.

Se abrirmos um arquivo de código de máquina em um editor de texto, veríamos um emaranhado de caracteres sem sentido. É possível ter acesso ao código de máquina em formato hexadecimal por meio de softwares adequados.

O **código objeto** é a saída de um processo de compilação e trata-se de uma parte do código de máquina que ainda não foi vinculado em um programa completo por meio de um **linker**.

# Abrindo um Código de Maquina

Ao tentarmos abrir um código de máquina em um editor de texto comum visualizamos algo sem sentido como a sequência de caracteres abaixo:

# Abrindo um Código de Maquina

Ao tentarmos abrir um código de máquina em um editor de texto comum visualizamos algo sem sentido como a sequência de caracteres abaixo:

```
MZÀ? $Pÿv?èŠÿ]Ë3ÀP, ?F?ë?fF??, ?< uè2Àëä?Àt?Ba
Àu?C†à2Àùä?¬I, "t??"<\u?€<"u?¬I?öÃ□□é?îY?Ê. <å‰.
?€?~?ä?‰v, ?vüÿv?ÿv?□?èÅ?fÄ?ÿvþÿvü?èüêYY< V?< F
|?ë?Rÿvþÿvü?èWífÄ?< å]ËU< ifìHVW< ~?< F
```

Porém, é possível ter acesso ao código de máquina utilizando editores próprios que apresentam o código em hexadecimal, como o exemplo que segue no *frame* seguinte.

# Compilador

```
C:\Utility>debug v.exe  
-d 0 100  
0E3D:0000 CD 20 FF 9F 00 9A F0 FE .....0.....  
0E3D:0010 F0 07 17 03 F0 07 DF 07 .....  
0E3D:0020 FF FF FF FF FF FF FF .....L.  
0E3D:0030 D0 0C 14 00 18 00 3D 0E .....=.....  
0E3D:0040 05 00 00 00 00 00 00 00 .....  
0E3D:0050 CD 21 CB 00 00 00 00 00 ..!.....  
0E3D:0060 20 20 20 20 20 20 20 20 .....  
0E3D:0070 20 20 20 20 20 20 20 20 .....  
0E3D:0080 00 0D 76 2E 65 78 65 0D ..v.exe.D0WS\sys  
0E3D:0090 74 65 6D 33 32 5C 64 6F tem32\dosx..da r  
0E3D:00A0 65 64 65 20 28 63 61 72 ede (carregar an  
0E3D:00B0 74 65 73 20 64 6F 20 64 tes do dosx.exe)
```

# Compilador

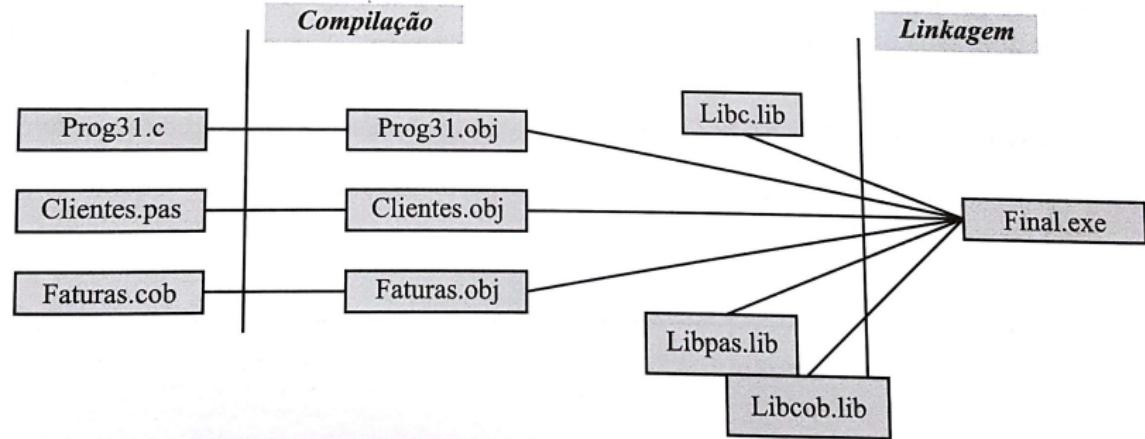


Figura: Diagrama (funcionamento de uma compilador).

# Compilador

Alguns autores citam linguagens compiladas em que a tradução do código gera código em C.

## **Maiores detalhes na referência abaixo**

Cooper, Torczon. Engineering a Compiler (em inglês). San Francisco: Morgan Kaufmann, 2003. p. 2. ISBN 1-55860-698-X.

# Compilador

## Importante

É importante não confundir **compilador** com **tradutor** ou **filtro** que também pode ser chamado de **conversor de linguagem**.

O conversor de linguagem é responsável por converter o código de uma linguagem de alto nível para o código de um outra linguagem de médio/alto nível.

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

# Compilador



Figura: Grace Hopper.

O primeiro compilador foi escrito por Grace Hopper no ano de 1952 para a linguagem de programação A-0.

Grace Hopper foi analista de sistemas da Marinha dos Estados Unidos. Ela também criou o primeiro compilador para a linguagem COBOL.

**Curiosidade:** É atribuído à Grace Hopper o termo **bug** utilizado para designar uma falha no código fonte.

# Compilador



Figura: Grace Hopper.

Grace Hopper é graduada em matemática e física em 1928 e em 1930 concluiu seu mestrado na Yale University. Em 1934, na mesma Universidade, ela obteve o seu PhD em matemática.

# Compilador

Muitos compiladores incluem um **pré-processador** que é um programa separado mas invocado pelo compilador antes do início do processo de tradução.

Normalmente é pre-processador responsável por mudanças no código fonte destinadas de acordo com decisões tomadas em tempo de compilação.

Em programas em C há diversas diretivas para inclusão de novos códigos disponíveis em bibliotecas ou código a parte escrito pelo programador que é informado sua existência por meio de diretivas para o pre-processador.

# Compilador

**Exemplo:** É o pré-processador que substitui os comentários do código fonte por espaços em branco. Ou seja, o compilador não “enxerga” nenhum comentário.

**Exemplos de linguagens compiladas:** C, C++, Fortran, Object-C, Ocaml, BASIC, COBOL, Ada, D, entre outras.

## O que é um interpretador?

**Resposta:** Interpretadores são programas de computador que tem a finalidade de ler o código-fonte de uma linguagem de programação interpretada e o converte em um código executável.

## O que é um interpretador?

**Resposta:** Interpretadores são programas de computador que tem a finalidade de ler o código-fonte de uma linguagem de programação interpretada e o converte em um código executável.

## Seu funcionamento pode ser, em geral, de duas formas:

- O interpretador lê linha-por-linha e converte o código fonte em código objeto (ou bytecode) a medida que vai executando o programa.

# Interpretador

## O que é um interpretador?

**Resposta:** Interpretadores são programas de computador que tem a finalidade de ler o código-fonte de uma linguagem de programação interpretada e o converte em um código executável.

## Seu funcionamento pode ser, em geral, de duas formas:

- O interpretador lê linha-por-linha e converte o código fonte em código objeto (ou bytecode) a medida que vai executando o programa.
- O interpretador converte o código fonte por inteiro e depois o executa.

## O que é um interpretador?

**Resposta:** Interpretadores são programas de computador que tem a finalidade de ler o código-fonte de uma linguagem de programação interpretada e o converte em um código executável.

## Seu funcionamento pode ser, em geral, de duas formas:

- O interpretador lê linha-por-linha e converte o código fonte em código objeto (ou bytecode) a medida que vai executando o programa.
- O interpretador converte o código fonte por inteiro e depois o executa.

**Exemplos de linguagens interpretadas:** R, Perl, Python, Haskell, Lua, Ruby, Lisp, JavaScript, entre outras.

# Programando em C

Na linguagem C existe uma função em que são colocadas todas as instruções que queremos que sejam executadas. Essa função chama-se `main()`, e todo código a executar é colocado entre `{}`.

Ao conjunto de código existente dentro de `{}` chamaremos de **bloco**.

**Exemplo:** Compile e execute seu primeiro programa utilizando a IDE Code::Blocks.

```
main()
{
}
```

# Programando em C

Como podemos observar, este programa faz uma das coisas que mais gosto na vida: não fazer nada. 😊

# Programando em C

Como podemos observar, este programa faz uma das coisas que mais gosto na vida: não fazer nada. 😊

Observe com atenção cada linha do programa. A primeira linha é composta pela palavra chave `main` que delimita o local onde todos os programas em C começam.

# Programando em C

Como podemos observar, este programa faz uma das coisas que mais gosto na vida: não fazer nada. 😊

Observe com atenção cada linha do programa. A primeira linha é composta pela palavra chave `main` que delimita o local onde todos os programas em C começam.

**Observação:** Para indicar que `main` é uma função, tal palavra chave é seguida com parênteses - `main()` pois **em C qualquer função tem que ser seguida por parênteses**.

## Importante

Os parênteses sem mais nada após o nome da função indicam que a função não recebe qualquer informação do mundo exterior.

# Programando em C

## Importante

Os parênteses sem mais nada após o nome da função indicam que a função não recebe qualquer informação do mundo exterior.

## Mais Importante Ainda

C é uma linguagem **Case Sensitive**, isto significa que C faz diferenciação entre maiúscula e minúscula. Assim, não são a mesma coisa escrever `main()`, `Main()`, `MAIN()`, `mAiN()`, etc.

# Programando em C

## Importante

Os parênteses sem mais nada após o nome da função indicam que a função não recebe qualquer informação do mundo exterior.

## Mais Importante Ainda

C é uma linguagem **Case Sensitive**, isto significa que C faz diferenciação entre maiúscula e minúscula. Assim, não são a mesma coisa escrever `main()`, `Main()`, `MAIN()`, `mAiN()`, etc.

**Dica:** Todas as funções em C são escritas em letra minúscula, e só se deve utilizar letras maiúsculas quando desejamos utilizar variáveis, mensagens ou funções escritas por nós.

# Programando em C

No exemplo anterior, o código não executa nada uma vez que não há código dentro do bloco de instruções composto pela função `main()`.

**Nota:** É possível que o seu compilador tenha apresentado um *WARNING* com uma mensagem semelhante a “**Function should return value**” ou “**main: no return value**”. Isso se deve ao fato de que a princípio estamos utilizando um compilador para C++.  
Duas possíveis soluções caso isso ocorra são:

# Programando em C

No exemplo anterior, o código não executa nada uma vez que não há código dentro do bloco de instruções composto pela função `main()`.

**Nota:** É possível que o seu compilador tenha apresentado um *WARNING* com uma mensagem semelhante a “**Function should return value**” ou “**main: no return value**”. Isso se deve ao fato de que a princípio estamos utilizando um compilador para C++.  
Duas possíveis soluções caso isso ocorra são:

- ① Ignorar o aviso. *WARNING* não é erro mas devemos prestar atenção nos avisos para evitarmos problemas.

# Programando em C

No exemplo anterior, o código não executa nada uma vez que não há código dentro do bloco de instruções composto pela função `main()`.

**Nota:** É possível que o seu compilador tenha apresentado um *WARNING* com uma mensagem semelhante a “**Function should return value**” ou “**main: no return value**”. Isso se deve ao fato de que a princípio estamos utilizando um compilador para C++.  
Duas possíveis soluções caso isso ocorra são:

- ① Ignorar o aviso. *WARNING* não é erro mas devemos prestar atenção nos avisos para evitarmos problemas.
- ② Coloque a palavra chave `void` antes da função `main()` na forma que segue.

# Programando em C

```
void main()
{
}
```

# Programando em C

```
void main()
{
}
```

Mais tarde nós saberemos o porquê do uso de void.

# Programando em C

```
void main()
{
}
```

Mais tarde nós saberemos o porquê do uso de void.

Vamos agora escrever um novo programa que faz algo mais interessante do que nada. Considere o exemplo que segue:

# Programando em C

## Exemplo:

```
1: #include <stdio.h>
2: main()
3: {
4:     printf("Hello Mundo Cruel");
5: }
```

# Programando em C

## Exemplo:

```
1: #include <stdio.h>
2: main()
3: {
4:     printf("Hello Mundo Cruel");
5: }
```

Esse código é em tudo semelhante ao anterior, com a exceção da existência de uma linha de código entre as chaves.

# Programando em C

## Exemplo:

```
1: #include <stdio.h>
2: main()
3: {
4:     printf("Hello Mundo Cruel");
5: }
```

Esse código é em tudo semelhante ao anterior, com a exceção da existência de uma linha de código entre as chaves.

A linha 4 é responsável pela apresentação da mensagem que desejamos imprimir: “Hello Mundo Cruel”.

## Importante

Sempre que queremos tratar conjunto de caracteres temos que colocá-los entre aspas, para que sejam considerados como um todo, isto é, devemos fazer "Hello Mundo Cruel" para o exemplo acima.

# Programando em C

## Importante

Sempre que queremos tratar conjunto de caracteres temos que colocá-los entre aspas, para que sejam considerados como um todo, isto é, devemos fazer "Hello Mundo Cruel" para o exemplo acima.

## A função printf()

Uma das funções que permite a escrita na tela é a função `printf()` = `print` + `formatado`. Como trata-se de uma função, há a necessidade de colocar os parênteses.

## Importante

Sempre que queremos tratar conjunto de caracteres temos que colocá-los entre aspas, para que sejam considerados como um todo, isto é, devemos fazer "Hello Mundo Cruel" para o exemplo acima.

## A função printf()

Uma das funções que permite a escrita na tela é a função `printf()` = `print` + `formatado`. Como trata-se de uma função, há a necessidade de colocar os parênteses.

Dentro do parênteses é feita a comunicação com a função. Nesse exemplo, passamos a `string` (cadeia de caracteres) que queremos que seja escrita - `printf("Hello Mundo Cruel")`.

## Muito Importante

## Muito Importante

- ① Em C, cada instrução deve ser terminada com um ponto-e-vírgula (;), obtendo-se assim a linha 4 do programa -  
`printf("Hello Mundo Cruel");`

## Muito Importante

- ① Em C, cada instrução deve ser terminada com um ponto-e-vírgula (;), obtendo-se assim a linha 4 do programa -  
`printf("Hello Mundo Cruel");`
- ② Preste muita atenção e perceba que o caractere " (aspas) é um **único** caractere e não pode ser substituído pelo caractere aspas simples ' nem muito menos por duas aspas simples ''.

## Muito Importante

- ① Em C, cada instrução deve ser terminada com um ponto-e-vírgula (;), obtendo-se assim a linha 4 do programa -  
`printf("Hello Mundo Cruel");`
- ② Preste muita atenção e perceba que o caractere " (aspas) é um **único** caractere e não pode ser substituído pelo caractere aspas simples ' nem muito menos por duas aspas simples ''.

**Observação:** A linguagem C não possui mecanismos de **Entrada/Saída (Input/Output)**. Dessa forma, precisamos recorrer a um conjunto de funções em **bibliotecas de funções (bibliotecas)**.

# Programando em C

Dessa forma, precisamos adicionar à linguagem C um conjunto de funções que, “por defeito”, ela não nos proporciona.

# Programando em C

Dessa forma, precisamos adicionar à linguagem C um conjunto de funções que, “por defeito”, ela não nos proporciona.

Para ter acesso a esse conjunto de funções teremos que incluir a sua definição no nosso código fonte.

# Programando em C

Dessa forma, precisamos adicionar à linguagem C um conjunto de funções que, “por defeito”, ela não nos proporciona.

Para ter acesso a esse conjunto de funções teremos que incluir a sua definição no nosso código fonte.

No exemplo anterior fazemos isso com a linha `#include <stdio.h>`.

# Programando em C

# Programando em C

**A linha #include <stdio.h> é C?**

**Resposta:** A linha `#include <stdio.h>` **não** é C mas uma diretiva que indica ao pré-processador que deverá ser adicionado antes da compilação um arquivo existente em alguma parte no disco do computador. Tal arquivo nesse caso é **stdio.h**.

# Programando em C

**A linha #include <stdio.h> é C?**

**Resposta:** A linha `#include <stdio.h>` **não** é C mas uma diretiva que indica ao pré-processador que deverá ser adicionado antes da compilação um arquivo existente em alguma parte no disco do computador. Tal arquivo nesse caso é **stdio.h**.

Esses arquivos são normalmente chamados de arquivos cabeçalhos e tem extensão .h, pois não têm código, mas apenas os cabeçalhos (**headers**) das funções que apresentam. Normalmente são chamados de **header files**.

# Programando em C

**A linha #include <stdio.h> é C?**

**Resposta:** A linha `#include <stdio.h>` **não** é C mas uma diretiva que indica ao pré-processador que deverá ser adicionado antes da compilação um arquivo existente em alguma parte no disco do computador. Tal arquivo nesse caso é **stdio.h**.

Esses arquivos são normalmente chamados de arquivos cabeçalhos e tem extensão .h, pois não têm código, mas apenas os cabeçalhos (**headers**) das funções que apresentam. Normalmente são chamados de **header files**.

`#include <stdio.h>` = adiciona o arquivo **stdio.h** ao meu programa **exatamente nesta posição**.

# Programando em C

**Por que não colocamos ; (ponto-e-vírgula) depois de  
#include <stdio.h>?**

**Resposta:** Porque, como foi mencionado anteriormente,  
`#include <stdio.h>` é uma diretiva do pré-processador. Dessa forma, `#include <stdio.h>` não é um comando de C.

**Nota:** Não colocamos ; (ponto-e-vírgula) ao final de instruções que não são de C.

O arquivo `stdio.h` permite o acesso a todas as funções de Entrada/Saída que necessitamos, em que **stdio** significa **standard input/output**.

# Programando em C

O programa que imprime na tela a mensagem "Hello Mundo Cruel" poderia ser programado de uma forma diferente e equivalente da forma que segue.

**Exemplo:** Dividindo a mensagem em mais de um printf.

```
1: #include <stdio.h>
2: main()
3: {
4:     printf("Hello");
5:     printf(" ");
6:     printf("Mundo");
7:     printf(" ");
8:     printf("Cruel");
9: }
```

# Programando em C

O compilador de C é, em geral, liberal no que diz respeito à forma como escrevemos (**identamos**) o código. Este pode ser escrito de modo que cada programador possa entender.

# Programando em C

O compilador de C é, em geral, liberal no que diz respeito à forma como escrevemos (**identamos**) o código. Este pode ser escrito de modo que cada programador possa entender.

**Exemplo:** Compile o código C que segue abaixo.

```
#include <stdio.h>
main      ( )  {
    printf(
"Hello"
);
    printf(  "Mundo" )
;    printf(      "Cruel" )
; }
```

# Programando em C

**Observação:** Espaços em branco são ignorados pelo compilador de C.

Nos programas que escrevem a frase “Hello Mundo Cruel” apresentados anteriormente observa-se que o cursor fica posicionado imediatamente após a palavra “Cruel”. Isso ocorre pelo fato de o programador não ter mandado o cursor mudar de linha após a escrita da mensagem.

Normalmente as linguagens de programação apresentam funções ou instruções distintas para realizar uma escrita na tela seguida ou não de uma nova linha. Em C a filosofia é outra.

# Programando em C

**Como podemos mudar de linha ao escrever uma mensagem?**

**Resposta:** Tradicionalmente a mudança de linha é denominada

**New Line**, e em C é representada pelo símbolo \n.

**Exemplo:** Considere o exemplo que segue.

```
1: #include <stdio.h>
2: void main()
3: {
4:     printf("Hello Mundo Cruel\n");
5: }
```

# Programando em C

**Observação:** O caractere especial *New Line* representado por \n é um caractere como qualquer outro. Assim, poderemos utilizar \n quantas vezes acharmos necessário. Nesse caso, poderíamos fazer:

**Exemplo:** Considere o exemplo que segue:

```
1: #include <stdio.h>
2: void main()
3: {
4:   printf("Hello\n\n Mundo\nCruel");
5: }
```

# Programando em C

**Exercício:** Escreva um programa em C que possui a seguinte saída:

```
C  
eh uma otima linguagem. Eu serei  
um grande programador em  
C!
```

# Programando em C

**Exercício:** Escreva um programa em C que possui a seguinte saída:

```
C  
eh uma otima linguagem. Eu serei  
um grande programador em  
C!
```

**Uma solução:**

```
1: #include <stdio.h>  
2: void main()  
3: {  
4:     printf("C\n");  
5:     printf("eh uma otima linguagem. Eu serei");  
6:     printf("um grande programado em");  
7:     printf("C!");  
8: }
```

# Programando em C

**Exercício:**

# Programando em C

**Exercício: Outra solução utilizando apenas um printf()**

```
#include <stdio.h>
void main()
{
    printf("C\neh uma otima linguagem.
    Eu seirei\um grande programador em\nC!");
}
```

# Programando em C

Suponha agora que queremos escrever a seguinte mensagem na tela do computador.

# Programando em C

Suponha agora que queremos escrever a seguinte mensagem na tela do computador.

**Hoje eh um dia "LINDO" com uma aula linda!!!**

# Programando em C

Suponha agora que queremos escrever a seguinte mensagem na tela do computador.

**Hoje eh um dia "LINDO" com uma aula linda!!!**

**Exemplo (com erro de compilação):** Muitos seriam tentados a pensar no programa que segue.

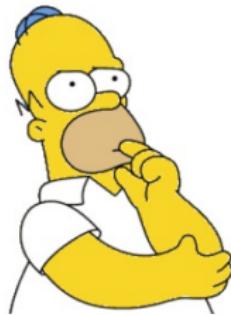
```
1: #include <stdio.h>
2: main()
3: {
4:     printf("Hoje eh um dia "LINDO" com uma aula
5:             linda!!!");
6: }
```

## Por que do erro do código acima?

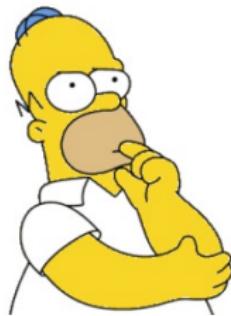
**Resposta:** O erro ocorre porque até a primeira ocorrência do símbolo " a linguagem C entende que estamos escrevendo "Hoje eh um dia ". Porém existe algo a mais dentro da função printf() que não é (*string*).

**Nota:** Chamamos de **string** uma cadeia de caracteres. Strings em C é definida entre aspas dupla (ex. "minha string").

# Programando em C



**Se as aspas servem para delimitar uma string, como poderemos escrever uma frase (string) em que as aspas sejam um caractere não delimitador?**



**Se as aspas servem para delimitar uma string, como poderemos escrever uma frase (string) em que as aspas sejam um caractere não delimitador?**

**Resposta:** Coloque \ antes das aspas que se pretende escrever na string. Assim, as aspas serão tratadas como caracteres normais.

# Programando em C

**Exemplo:** Para que possamos escrever a frase corretamente na tela do computador, considere o programa abaixo.

```
1: #include <stdio.h>
2: void main()
3: {
4:     printf("Hoje eh um dia \"LINDO\" com uma aula
5:             linda!!!");
6: }
```

# Programando em C

Table: O caractere especial \.

\7	<i>Bell</i> (sinal sonoro do computador)
\a	<i>Bell</i> (sinal sonoro do computador)
\b	<i>BackSpace</i>
\n	<i>New Line</i> (mudança de linha)
\r	<i>Carroage Return</i>
\t	Tabulação Horizontal
\v	Tabulação Vertical
\\"	Caractere \
'	Caractere aspas simples '
"	Caractere aspas dupla "
?	Caractere ? (ponto de interrogação)
%%	Caractere %

# Programando em C

**Exercício:** Crie alguns programas em C que faz uso de alguns caracteres especiais.

É muito importante a prática de se comentar muito bem um código em que escrevemos. Nunca se sabe quando teremos a necessidade de reprogramar o código.

É comum em um código não comentado esquecermos o que foi feito em um código fonte programado a muito tempo.

**Observação:** O bom programador segue “sempre” boas práticas de programação. Comentar bem um código por mais simples que ele seja é uma boa prática de programação.

# Programando em C

## Como são feitos comentários em C?

**Resposta:** Comentários em C são colocados entre /\* \*/ (*/\* AQUI É UM COMENTÁRIO \*/*).

**Alguns exemplos de cabeçalhos de programas escritos em C são:**

```
/* programa.c: Empresa Comentários e CIA */
/* AUTOR: Pedro Rafael */
/* DATA: 02/02/2017 */
```

Acima temos três comentários em que cada um deles estão entre /\* \*/. Porém, poderíamos ter feito:

# Programando em C

```
/* programa.c: Empresa Comentários e CIA
 * AUTOR: Pedro Rafael
 * DATA: 02/02/2017
 */
```

ou então:

```
*****  
* programa.c: Empresa Comentários e CIA  *  
* AUTOR: Pedro Rafael                      *  
* DATA: 02/02/2017                          *  
*****/
```

**Observação:** Os asteriscos horizontais ou verticais **não** tem nenhum significado especial. Eles apenas fazem parte do que queremos comentar.

# Programando em C

## Importante

Os comentários podem também serem colocados dentro de expressões ou instruções.

**Exemplo:** Poderíamos fazer:

```
printf("Bom dia querido professor, nós o adoramos!"  
/* É mentira :P */; /* Notar o ponto-ponto e vírgula  
*/
```

**Dica:** Particularmente, não acho uma boa prática de programação comentar no interior de instruções ou expressões. Tal postura tornará o código difícil de ser entendido.

# Programando em C

## Importante

Os compiladores de C **não** permitem, em geral, comentários dentro de comentários.

**Exemplo:** Comentários dentro de comentários não são permitidos, em geral, em C.

```
/* Meu primeiro comentário.  
/* Meu segundo comentário sobre o primeiro  
   comentário. */  
*/
```

**Nota:** Alguns compiladores permitem o uso de comentários dentro de comentários. Porém, tais opção não é reconhecida pelo ANSI.

# Programação Aplicada à Estatística

Pedro Rafael Diniz Marinho

Universidade Federal da Paraíba

2021.2

# Tipos de Dados em C



Sempre que abrimos nossa geladeira nos deparamos com uma enorme variedade de recipientes para cada tipo de produto: sólido, líquido, regulares, irregulares, etc.

Em C não é diferente. Precisamos conhecer aquilo que queremos armazenar e só então selecionar os melhores recipientes para a tarefa.

# Tipos de Dados em C

Existem quatro tipos de dados básicos em C. São ele:

# Tipos de Dados em C

Existem quatro tipos de dados básicos em C. São ele:

- ① **char;**

# Tipos de Dados em C

Existem quatro tipos de dados básicos em C. São ele:

- ① **char;**
- ② **int;**

# Tipos de Dados em C

Existem quatro tipos de dados básicos em C. São ele:

- ① **char;**
- ② **int;**
- ③ **float;**

# Tipos de Dados em C

Existem quatro tipos de dados básicos em C. São eles:

- ① **char;**
- ② **int;**
- ③ **float;**
- ④ **double.**

A medida que formos adiantando o assunto entenderemos detalhadamente sobre cada um dos quatro tipos básicos de dados.

**Observação:** Existe ainda um outro tipo - tipo **Ponteiro** - que poderá ser considerado um tipo básico. Trataremos desse tipo quando estivermos um pouco mais adiantados no curso.

# Variáveis

Sempre que temos o interesse de armazenar um valor que, por algum motivo, não seja fixo, devemos fazê-lo utilizando variáveis.

## O que é um variável em C?

**Resposta:** Uma **variável** é nada mais que um nome que damos a uma determinada posição na memória para conter um valor de um determinado tipo.

Como o próprio nome indica, o valor contido em uma variável pode variar ao longo da execução de um programa.

## Muito Importante

Uma variável deve ser **sempre** definida **antes** de ser utilizada. O compilador normalmente indica a não declaração de uma variável na mensagem de aviso contendo o erro de compilação.

**Na definição de uma variável, precisamos especificar ao compilador qual o tipo de dado que fica atribuído ao nome que indicamos para essa variável.**

# Variáveis

## Declarando uma Variável

A definição de variáveis é feita utilizando a seguinte sintaxe:

```
tipo var1 [, var2, ..., varn] ;
```

**Observação:** Tudo que está dentro de colchetes é opcional e os colchetes não faz parte da declaração. Por exemplo, se quisermos declarar duas variáveis do mesmo tipo, podemos fazer: tipo var<sub>1</sub>, var<sub>2</sub>;. O mesmo raciocínio deverá ser seguido para três ou mais variáveis.

# Variáveis

**Exemplo:** O trecho de código abaixo ilustra a declaração dos tipos básicos de variáveis em C.

```
int i;      /* i é uma variável do tipo inteiro */
char ch1, novo_char; /* ch1 e novo_char é uma variável
do tipo char */
float pi, raio, perimetro; /* Estas são variáveis do
tipo float */
double total, k123; /* Aqui estão as variáveis do
tipo double */
```

# Variáveis

Sim, eu sou chato 😊 e vou repetir novamente: **Uma variável deverá ser declarada antes de sua utilização e antes de qualquer instrução. O esquema abaixo ilustra isso:**

```
main()
{
    Declaração de variáveis;

    Instrução 1;
    Instrução 2;
}
```

# Variáveis

As variáveis são armazenadas em um endereço de memória do computador. Por meio do nome da variável é que acessamos a posição na memória que armazena o tipo de dado básico que armazenamos.

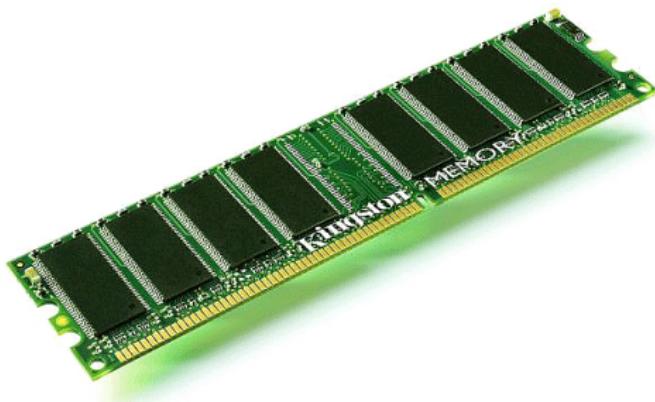


Figura: Memória de um computador digital.

# Variáveis

É muito mais conveniente e simples referenciar os endereços da memória por meio de nomes de variáveis. O tipo que está associado ao nome da variável indica a quantidade de *bytes* que serão utilizados para guardar um valor nessa variável.

**Para nomear as variáveis, é conveniente observar um conjunto de regras que serão listadas abaixo:**

# Variáveis

É muito mais conveniente e simples referenciar os endereços da memória por meio de nomes de variáveis. O tipo que está associado ao nome da variável indica a quantidade de *bytes* que serão utilizados para guardar um valor nessa variável.

**Para nomear as variáveis, é conveniente observar um conjunto de regras que serão listadas abaixo:**

- O nome de uma variável pode ser construído por letras do alfabeto (minúsculas ou maiúsculas), dígitos (de 0 a 9) e ainda pelo caractere underscore (\_).

# Variáveis

É muito mais conveniente e simples referenciar os endereços da memória por meio de nomes de variáveis. O tipo que está associado ao nome da variável indica a quantidade de *bytes* que serão utilizados para guardar um valor nessa variável.

**Para nomear as variáveis, é conveniente observar um conjunto de regras que serão listadas abaixo:**

- O nome de uma variável pode ser construído por letras do alfabeto (minúsculas ou maiúsculas), dígitos (de 0 a 9) e ainda pelo caractere underscore (\_).
- O primeiro caractere **não** poderá ser um dígito. Terá que ser uma letra ou o caractere underscore.

# Variáveis

# Variáveis

- O uso do caractere underscore no início de uma variável é desaconselhável mas não é errado.

# Variáveis

- O uso do caractere underscore no início de uma variável é desaconselhável mas não é errado.
- Uma variável jamais poderá ter o nome de uma palavra reservada em C como main, int, float, char, double, if, while, else, return, goto, entre diversas outras.

# Variáveis

- O uso do caractere underscore no início de uma variável é desaconselhável mas não é errado.
- Uma variável jamais poderá ter o nome de uma palavra reservada em C como main, int, float, char, double, if, while, else, return, goto, entre diversas outras.
- Lembre-se que a linguagem C é *Case Sensitive*. Assim, int a é diferente de declarar int A.

# Variáveis

- O uso do caractere underscore no início de uma variável é desaconselhável mas não é errado.
- Uma variável jamais poderá ter o nome de uma palavra reservada em C como main, int, float, char, double, if, while, else, return, goto, entre diversas outras.
- Lembre-se que a linguagem C é *Case Sensitive*. Assim, int a é diferente de declarar int A.

**Observação:** Evite a declaração de variáveis utilizando caracteres acentuados (ã, á, à, ó, etc.). A maioria dos compiladores não aceita tais caracteres como admissíveis.

# Variáveis

**Exemplo:** Declarações de variáveis em C.

```
int idade; /* Correto */  
int Num_Cliente; /* Correto */  
float a1b2c3; /* Correto */  
float 7a2b3c; /* Incorreto */  
char float; /* Incorreto */  
double vinte%; /* Incorreto */  
char sim?não; /* Incorreto */  
int _alfa; /* Correto */  
int _123; /* Correto */  
char num, NUM; /* Correto */
```

**Dica:** Na segunda declaração vemos um bom uso do caractere underscore que foi utilizado para tornar mais legível a variável.

# Variáveis

**Quantos caracteres podemos utilizar para definir uma variável?**

**Resposta:** O número de caracteres irá depender do compilador de C utilizado. Normalmente é permitido nomes de variáveis com até 32 caracteres (ou mais).

**Algumas boas práticas de programação são:**

- O nome de uma variável deve nos lembrar do que ela de fato armazena;
- O nome de uma variável não deve ser todo escrito em maiúscula. Normalmente a maioria dos programadores definem constantes dessa forma.
- Caso o nome de uma variável utilize mais de uma palavra, utilize a dica do *frame* anterior.

# Variáveis

**Nota:** Quando uma variável é declarada fica sempre com um valor, qual resulta do estado aleatório de bits que a constituem. Isto é, uma variável declarada sempre possui um lixo no seu conteúdo.

Porém, é sempre conveniente inicializar uma variável com um valor através de uma operação de atribuição. Assim, ao realizar uma atribuição, o valor antigo na memória ao qual a variável ocupa é eliminado, ficando nela o novo valor atribuído.

**A operação de atribuição obedece a seguinte sintaxe:**

variável = expressão ou valor atribuído;

# Variáveis

A atribuição de valores em C é realizada através do sinal = e a variável **SEMPRE** é colocada ao **lado esquerdo** da atribuição.

# Variáveis

A atribuição de valores em C é realizada através do sinal = e a variável **SEMPRE** é colocada ao **lado esquerdo** da atribuição.

## Exemplo:

```
int num; /* Declaração da variável num */  
num = -17; /* num recebe o valor -17 */
```

# Variáveis

A atribuição de valores em C é realizada através do sinal = e a variável **SEMPRE** é colocada ao **lado esquerdo** da atribuição.

## Exemplo:

```
int num; /* Declaração da variável num */  
num = -17; /* num recebe o valor -17 */
```

**Observação:** Podemos fazer uma atribuição à variável no momento de sua declaração.

# Variáveis

A atribuição de valores em C é realizada através do sinal = e a variável **SEMPRE** é colocada ao **lado esquerdo** da atribuição.

## Exemplo:

```
int num; /* Declaração da variável num */  
num = -17; /* num recebe o valor -17 */
```

**Observação:** Podemos fazer uma atribuição à variável no momento de sua declaração.

## Exemplo:

```
int num = -17;  
int n1 = 3, n2 = 5;  
int a = 10, b, c = -123, d;
```

# Variáveis

**Observação:** Podemos passar o valor contido no endereço de memória de uma variável para o endereço de memória de outra variável.

**Exemplo:**

```
int a = 3, b = 7;  
a = b; /* a variável "a" conterá o valor 7 */
```

# Variáveis

**Nota:** Em C é possível atribuir o mesmo valor a várias variáveis.

**Exemplo:** Colocar o valor 5 nas variáveis a, b, c e d previamente declaradas:

a = 5;

b = 5;

c = 5

d = 5;

Seria mais conveniente fazer a = b = c = d = 5;. Tal declaração está correta.

**Observação:** As atribuições em C são realizadas da direita para a esquerda.

# Inteiros - int

As variáveis do tipo **int** são utilizadas para armazenar valores naturais (sem parte fracionária) positivos e negativos. Por exemplo, 2, 0, 7, -345, +115.

Table: Operações sobre números inteiros.

Operação	Descrição	Exemplo	Resultado
+	Soma	$21 + 4$	25
-	Subtração	$21 - 4$	17
*	Multiplicação	$21 * 4$	84
/	Divisão	$21 / 4$	5
%	Módulo	$21 \% 4$	1

# Inteiros - int

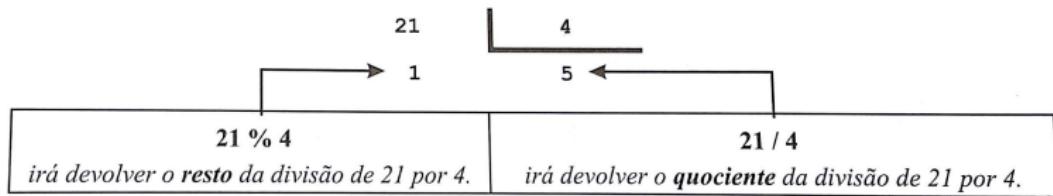
**Observação:** O **módulo** trata-se do resto da divisão inteira. O operador / fornece o quociente da divisão de dois inteiros. Já o operador % (módulo) fornecerá o resto da divisão de dois números inteiros.

## Muito Importante

Qualquer operação entre inteiros retorna sempre um inteiro. Nesse caso, por exemplo,  $21/4$  retornará não irá retornar o valor 5.25 como poderíamos pensar. Teremos nesse caso o retorno do valor 5.

# Inteiros - int

A imagem abaixo apresenta bem a diferença do uso dos operadores / e %.



# Inteiros - int

Consideremos com atenção o código no exemplo que segue:

## Exemplo:

```
1: #include <stdio.h> /* Essa linha não é código C */
2:
3: void main()
4: {
5:     int num = 123;
6:
7:     printf("O valor de num = %d e o
8:             valor seguinte = %d\n", num, num+1);
9: }
```

O que o programa imprime em tela?

## Inteiros - int

**Observação:** Sempre que estamos interessados em escrever o valor de um inteiro dentro de um **printf** devemos substituir o valor desse inteiro por um **formato de escrita**.

**Em geral, qual o formato de escrita de um inteiro em C?**

**Resposta:** O formato de escrita de um inteiro na função **printf()** é **%d**.

Devemos colocar **%d** na função **printf()** no lugar onde queremos substituir o valor armazenado na variável inteira. Ao final da string, especificamos de quais variáveis esses valores serão obtidos.

## Inteiros - int

Da mesma forma que existe uma função `printf()` para escrita de valores, existe uma função correspondente para a leitura de valores, está é a função `scanf()`.

**Exercício:** Implemente o código abaixo e explique cada linha de código do programa que segue.

```
1: #include <stdio.h>
2:
3: void main()
4: {
5:     int num;
6:     printf("Introduza um numero: ");
7:     scanf("%d", &num);
8:     printf("O numero introduzido foi %d\n", num);
9: }
```

## Função scanf

A função `scanf` = `scan` + `formatado` funciona de forma semelhante à função `printf`. Uma vez que ela foi implementada para a leitura de valores, a *string* inicial (primeiro argumento da função) deve conter apenas o formato das variáveis que queremos ler. **Exceto se a variável for uma string.**

Depois de especificados os formatos de leitura na *string*, devem ser colocadas todas as variáveis correspondentes pela ordem em que ocorrem os formatos, precedidas de um &.

**Observação:** O esquecimento do (“E” comercial) poderá gerar resultados inesperados. Preste atenção no seu uso!

# Inteiros - int

**Observação:** Na função `scanf` preste atenção que o tipo de leitura “%d” necessita que a variável `num` tenha o mesmo tipo que nesse caso é inteiro.

## Importante

A string enviada para a função `scanf` não deve conter outros caracteres que não sejam os caracteres indicadores de formato. Jamais coloque `\n` na *string* da função `scanf`.

# Inteiros e Variações

Os tipos de dados variam de arquitetura para arquitetura. No caso do tipo **int** são habituais os valores de 2 ou 4 *bytes*. Sendo assim, é importante saber qual a dimensão de um inteiro quando se desenvolve uma aplicação, caso contrário corremos o risco de tentar armazenar um valor em uma variável inteira com o número de *bytes* insuficiente.

## E como eu sei a dimensão de um int no meu computador?

**Resposta:** A linguagem C disponibiliza um operador denominado `sizeof`. Seu uso é da forma que segue:

```
sizeof <expressão> ou sizeof (<tipo de dados>)
```

**Nota:** <> não faz parte da sintaxe!

# Inteiros e Variações

**Exemplo:** O programa abaixo indica o número de **bytes** que ocupa um inteiro.

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("O tamanho em bytes de um inteiro = %d\n",
6:            sizeof(int));
7: }
```

**Exercício:** Crie um programa que apresente tamanho em número de *bytes* dos tipos de dados **int**, **char**, **float** e **double**.

# Inteiros e Variações

## Importante

O fato de o tamanho de um inteiro poder variar de arquitetura para arquitetura é algo preocupante, pois os limites das variáveis que armazenam um inteiro podem variar de maneira drástica. Isso reduz consideravelmente a portabilidade dos programas entre máquinas diferentes. Veja a tabela abaixo:

Tabela: Variação de inteiros pela quantidade de *bytes*.

Número de Bytes	Menor Valor	Maior Valor
2	-32768	32767
4	-2147483648	2147483647

# Inteiros e Variações

A forma que podemos garantir que o programa venha sempre utilizar a mesma quantidade de *bytes* independente da arquitetura é utilizando quatro prefixos distintos para melhor definição da característica da variável. São eles:

- ① **short** - Inteiro pequeno (2 *bytes*);
- ② **long** - Inteiro grande (4 *bytes*);
- ③ **signed** - Inteiro com sinal (números negativos e positivos);
- ④ **unsigned** - Inteiro sem sinal (apenas números positivos).

# Inteiros e Variações

Para que possamos garantir que o inteiro  $n$  utiliza apenas 2 *bytes* de memória, independentemente da arquitetura utilizada, devemos declarar a variável como

```
short int n; /* ou short n; */
```

Já para que possamos garantir de o inteiro  $n$  usa sempre 4 *bytes* de memória, independentemente da arquitetura utilizada, devemos declarar a variável como

```
long int n; /* ou long n; */
```

**Observação:** O prefixo **short** garante o tamanho mínimo do inteiro e o prefixo **long** garante o tamanho máximo.

# Inteiros e Variações

Tabela: Quantidade de *bytes* reservadas em diferentes tipos de inteiros.

Sistema (bits)	short int	int	long int
32 bits	2	2	4
64 bits	2	4	8

## Importante

O formato de leitura e escrita de variáveis inteiras short e long nas funções scanf e printf deve ser precedido dos prefixos **h** (short) e **l** (long).

# Inteiros e Variações

**Exercício:** Compile o programa e explique cada linha de código.

```
1: #include <stdio.h>
2:
3: void main()
4: {
5:     short int idade;      /* ou short idade */
6:     int montante;
7:     long int n_conta;    /* ou long n_conta */
8:
9:     printf("Qual a Idade? ");
10    scanf("%hd", &idade);
11    printf("Qual o montante a depositar? ");
12    scanf("%d", &montante);
13    printf("Qual o numero da conta? ");
```

# Inteiros e Variações

```
14:     scanf("%ld", &n_conta);
15:     printf("Uma pessoa de %hd anos depositou R$%d reais
16                 na conta %ld\n", idade, montante, n_conta);
```

## Prefixos `signed` e `unsigned`

Uma variável do tipo inteiro admite valores inteiros **positivos** e **negativos**. Caso se deseje que a variável contenha apenas valores positivos, devemos declarar com o prefixo **`unsigned`**.

# Uma pequena sequência de exercícios

**Exercício:** Crie um programa em C que solicite ao usuário para digitar 2 valores inteiros e coloque o primeiro valor digitado em uma variável **a** e o segundo valor digitado em uma variável **b**. Em seguida, o programa deverá imprimir em tela o conteúdo dessas variáveis.

# Uma pequena sequência de exercícios

**Exercício:** Crie um programa em C que solicite ao usuário para digitar 2 valores inteiros e coloque o primeiro valor digitado em uma variável **a** e o segundo valor digitado em uma variável **b**. Em seguida, o programa deverá imprimir em tela o conteúdo dessas variáveis.

**Exercício:** Cria um programa em C que solicite ao usuário para digitar 4 valores inteiros e armazene os valores nas variáveis **a**, **b**, **c** e **d**, respectivamente. O seu programa deverá retornar o resultado da seguinte expressão matemática:

$$(a + b + c) \times d.$$

# Uma pequena sequência de exercícios

**Exercício:** Construa um programa que realize uma conversão de valores de reais para dólares. Dessa forma, o usuário deverá ser solicitado do valor em reais e deverá imprimir em tela o valor equivalente em dólares americano. Considere que R\$ 1.00 reais equivale à \$ 0.30796 dólares. (**Dica:** Como nesse ponto sabemos definir variáveis reais, não há a necessidade de considerar apenas valores inteiros.)

# Uma pequena sequência de exercícios

**Exercício:** Construa um programa que realize uma conversão de valores de reais para dólares. Dessa forma, o usuário deverá ser solicitado do valor em reais e deverá imprimir em tela o valor equivalente em dólares americano. Considere que R\$ 1.00 reais equivale à \$ 0.30796 dólares. (**Dica:** Como nesse ponto sabemos definir variáveis reais, não há a necessidade de considerar apenas valores inteiros.)

**Exercício:** A condição física de uma pessoa pode ser medida com base no cálculo do Índice de Massa Corporal (IMC). O mesmo é calculado dividindo-se o peso desta pessoa em kg pelo quadrado sua altura em m. Escreva um programa que leia o peso em kg e a altura de uma pessoa em m, calcule e mostre o IMC. Se as entradas fossem 70.0kg para o peso 1.80m para altura então a saída seria aproximadamente 21.60.