

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Apresentação

- Professor:

**Bernardo Martins**

Bacharel em Ciência da Computação na UESB

Mestrando em Ciência da Computação no IME-USP

Desenvolvedor back end - nodejs, AWS

E-mail: [bernardomartinsf@ime.usp.br](mailto:bernardomartinsf@ime.usp.br)

# Apresentação

Bem-vindos ao curso!

Falem sobre vocês!

# Apresentação

- Quem nunca teve contato com programação?

# Apresentação

- Quem nunca teve contato com programação?
- Quem já teve contato com programação? Qual linguagem?

# Apresentação

- Quem nunca teve contato com programação?
- Quem já teve contato com programação? Qual linguagem?
- O que os motivou a participar deste curso?

# Objetivo

*"Público: Estudantes da área de Ciências Exatas, alunos da 3<sup>a</sup> série do ensino médio, e profissionais que desejam programar numa linguagem estruturada."*

# Objetivo

*"Público: Estudantes da área de Ciências Exatas, alunos da 3<sup>a</sup> série do ensino médio, e profissionais que desejam programar numa linguagem estruturada."*

- O objetivo ao final do curso é que sejam capazes de resolver problemas pela programação em linguagem estruturada (ANSI C).

# Objetivo

*"Público: Estudantes da área de Ciências Exatas, alunos da 3<sup>a</sup> série do ensino médio, e profissionais que desejam programar numa linguagem estruturada."*

- O objetivo ao final do curso é que sejam capazes de resolver problemas pela programação em linguagem estruturada (ANSI C).
- Fornecer uma base de conhecimento em computação, para que possam seguir no aprendizado de outras linguagens, paradigmas e tecnologias.

# Metodologia

- Durante as aulas:
  - Introdução ao conteúdo
  - Resolução de exercícios
  - Elaboração de código
  - Prática utilizando exemplos
- Fora das aulas:
  - Prática - Resolução de exercícios



# Calendário do curso

- Aulas diariamente, de segunda a sexta
- Horários: 19:00 as 21:00
- Início: 09/01/2023
- Fim: 17/02/2023
- Carga horária: 60h
- Uma aula de exercícios por semana
- As aulas serão gravadas e disponibilizadas

# Aulas de exercícios

- Dúvidas gerais sobre a matéria e exercícios
- Resolução de exercícios
- Contato: de preferência pelo ambiente virtual (fóruns ou mensagem particular)
- Ocorrerá ao menos uma vez por semana de acordo com a necessidade



# Avaliações

- Disponibilizadas no Moodle
- Permanecerão abertas por 24 horas
- Abertas no horário da aula
- Datas:
  - 27/01/2023 19:00 até 28/01/2023 19:00
  - 15/02/2023 19:00 até 16/02/2023 19:00

# Exercícios

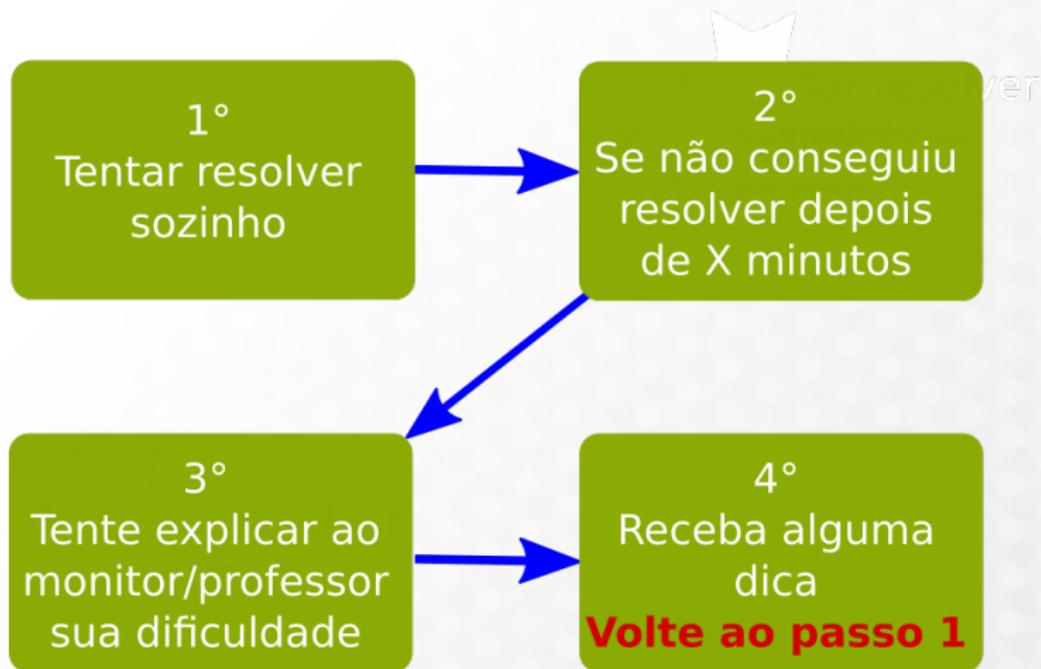
- Categorias:
  - Obrigatórios, compõem 50% da nota final
  - Fixação, para praticar
  - Aplicados, resolvidos durante a aula
  - Desafios, para aprofundar o conhecimento
- *Exercícios obrigatórios devem ser entregues no prazo*

# Importância de resolver sozinho os exercícios

- Existem trabalhos que comprovam a necessidade do esforço individual em problemas/exercícios para alcançar o entendimento.
- Metáfora do exercício físico: (**pipoca + sofá vs praticar + forma**)
- É essencial tentar resolver novos problemas

# Importância de resolver sozinho os exercícios

Adote o seguinte "algoritmo" (sequência de passos):



# Cálculo da nota final

$$NF = (NE \times 0.5) + (NA \times 0.5)$$

Onde:

$NF$  = Nota Final

$NE$  = Nota de Exercícios (obrigatórios)

$NA$  = Nota de Avaliações

# Requisitos para aprovação no curso

- **Nota Final  $\geq 5.0$**
- **Frequência  $\geq 75\%$** 
  - Contabilizada por meio do módulo “Registro de presença”  
[saw.atp.usp.br/mod/attendance/view.php?id=22657](http://saw.atp.usp.br/mod/attendance/view.php?id=22657)



# Tópicos do curso

- Arquitetura de computadores
- Lógica de programação
- Variáveis e comandos de entrada/saída
- Estruturas de seleção e desvio de fluxo
- Estruturas de repetição, vetores e ponteiros
- Funções e recursão

# Algumas ferramentas que serão usadas

- Ambiente virtual SAW (Sistema de Aprendizagem Web)
- VPL (Laboratório de Programação Virtual)
- iVProg (Programação Visual)
- BBB (BigBlueButton)

# Ambiente virtual

- SAW: <https://saw.atp.usp.br/course/view.php?id=493>
- No SAW, serão disponibilizados todos os recursos:
  - Materiais didáticos
  - Exercícios
  - Avaliações
  - Dúvidas
  - Comunicação
  - Área de videoconferência



# Avisos importantes

- Não pratique **plágio!**
- Caso enfrente dificuldades e dúvidas nos exercícios, use o fórum ou mensagem privada com o professor ou colega.
- Ao publicar suas dúvidas, não divulgue sua solução, indique qual exercício e explique seus questionamentos.
- Caso utilize alguma fonte para suas respostas, cite (site, livros, artigos, etc.).
- Leitura obrigatória do material **Plágio++ (cola)** no Moodle

## Demais avisos

- Assuntos administrativos devem ser tratados na Secretaria do Programa de Verão – email: [verao@ime.usp.br](mailto:verao@ime.usp.br).
- Para receber o certificado, é necessário satisfazer os critérios de aprovação no curso.

# Páginas importantes

- Curso
- Aulas
- Áreas de teste:
  - iVProg
  - C
- Frequência
- Fóruns:
  - Avisos Gerais
  - Dúvidas Gerais

# Dicas de ambientes para programação C

- VPL: Ambiente com avaliador automático que será utilizado durante o curso
- VSCode: Editor de código com diversas bibliotecas para facilitar o desenvolvimento
- JetBrains: Ambiente de desenvolvimento pago com soluções para cada espectro de linguagens (CLion, Webstorm, IntelliJ, etc)
- Vim e Emacs: Editores de código no termina com curva de aprendizado muito intensa

# Bibliografia

- DEITEL, H.M. e DEITEL, P.J., "C: Como Programar", 6a ed., Livros Técnicos e Científicos, 2011.
- ASCENCIO, A. F. G. e CAMPOS, E. A. V Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++. Prentice Hall. 2007.
- FORBELLONE, A. L. V. e EBERSPACHER, H. F. Lógica de Programação – A Construção de Algoritmos e Estrutura de Dados. 3a Edição. Prentice Hall, 2005.
- SOUZA, M. A. F. et al. Algoritmos e Lógica de Programação. C. Learning. 2008.



# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Computador

- O que é um computador?

# Computador

- O que é um computador?

"Um computador é um dispositivo capaz de realizar cálculos e tomar decisões lógicas em velocidades bilhões de vezes mais rápidas do que os seres humanos." Deitel & Deitel, 2007.

# Computador

- O que é um computador?

"Um computador é um dispositivo capaz de realizar cálculos e tomar decisões lógicas em velocidades bilhões de vezes mais rápidas do que os seres humanos." Deitel & Deitel, 2007.

- Computadores processam dados seguindo instruções
- Conjuntos de instruções são chamados **programas de computador**
- Programas de computador são chamados de **software**

# Computador - História

- Computar faz parte da essência humana
  - Astronomia
  - Enumeração
  - Impostos

# Computador - História



- Primeira máquina calculadora: "Pascaline"
- Somas e subtrações
- Blaise Pascal (1623 - 1662)
- Cálculo de impostos



IME-USP

# Computador - História

- Charles Babbage (1791 – 1871)

- Ada Lovelace (1815 – 1852)

- Colaborou com Babbage
- Primeira programadora
  - Subrotinas
  - Laços
  - Condicional



IME-USP

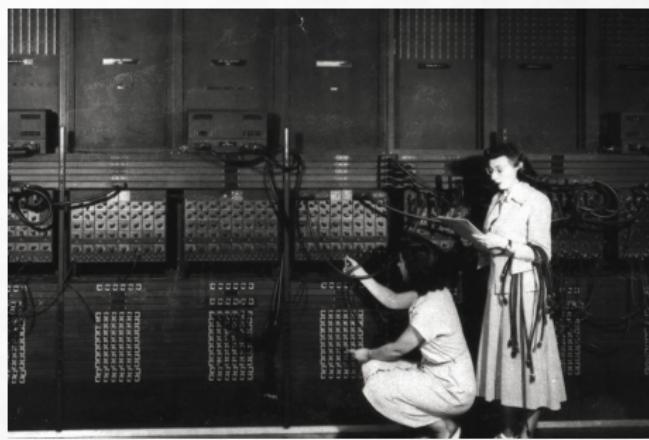
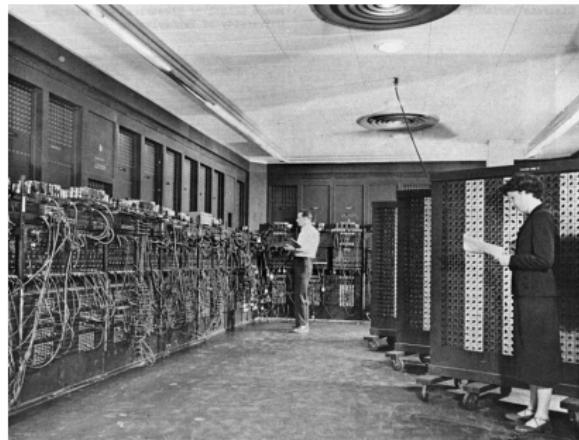
# Computador - História

- Alan Turing (1912 – 1954)
  - Máquina de Turing
- ENIAC (*Electronic Numerical Integrator and Computer*)
  - Primeiro computador digital eletrônico
  - 5.000 operações por segundo
  - Terminado após a guerra
  - Realizava cálculos de trajetória balística
  - Sistema operacional em cartões perfurados e programado por milhares de interruptores



# Computador - História

- ENIAC (*Electronic Numerical Integrator and Computer*)



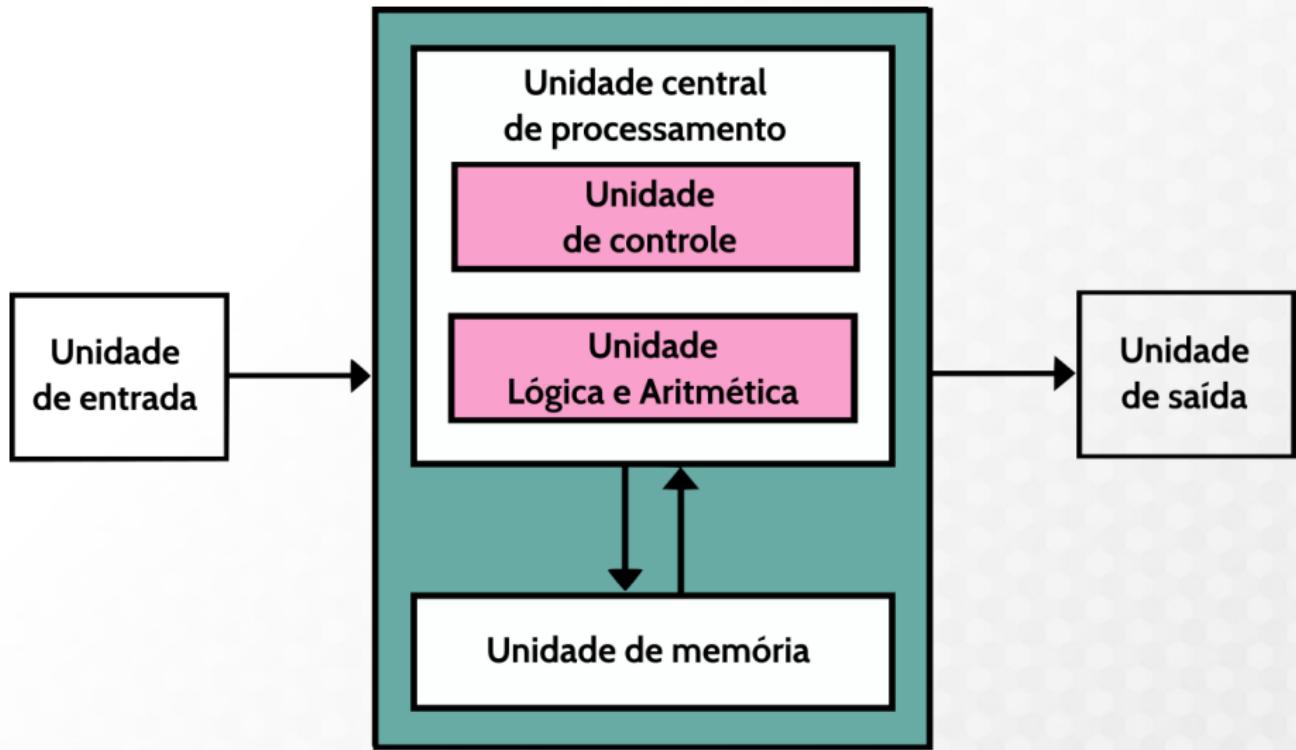
# Arquitetura de Von Neumann

"...é uma arquitetura de computador que se caracteriza pela possibilidade de uma máquina digital armazenar seus programas no mesmo espaço de memória que os dados, podendo assim manipular tais programas."

Leia mais em:

[http://pt.wikipedia.org/wiki/Arquitetura\\_de\\_von\\_Neumann](http://pt.wikipedia.org/wiki/Arquitetura_de_von_Neumann)

# Arquitetura de Von Neumann



- **Hardware:** parte física
  - Componentes físicos do computador
- **Software:** parte lógica
  - Programas a serem executados em um computador

**Software é o que você xinga, Hardware o que você chuta**

# Hardware e dispositivos

- A linguagem nativa do computador é codificada numericamente, de forma binária:
  - Bit: Pode assumir valores 0 ou 1.
  - Byte: Agrupamento de 8 bit em uma palavra.
  - Letras e símbolos são representados por números.

# Componentes físicos

## Dispositivos de entrada e dispositivos de saída



# Componentes físicos

- Processador

- Responsável geral pela execução de tarefas
- Registradores
- Unidade de controle
- Unidade lógica e aritmética



# Componentes físicos

- Memória

- Qualquer dispositivo de memória
- Memória RAM
- Memória ROM
- Memória cache (L1, L2 e L3)
- Disco rígido
- SSD



# Componentes físicos

- Memória
  - Memória **primária**
    - RAM, ROM e caches.
  - Memória **secundária**
    - Dispositivos de armazenamento em massa.



# Sistema Operacional (SO)

- Sistemas mais conhecidos:
  - Windows
  - OS X
  - Linux
- Visão *bottom-up*
  - Perspectiva do programador
  - Intermedia a comunicação entre programas e componentes físicos
- Visão *top-down*
  - Gerenciador de recursos



# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Introdução à Lógica

- **Lógica:** é a técnica de encadear pensamentos para atingir determinado objetivo;
- **Sequência Lógica:** são passos executados até atingir um objetivo ou solução de um problema;
- **Instruções:** regras ou normas definidas para a realização ou emprego de algo. É o que indica a um computador uma ação elementar a executar.



## O que é um algoritmo?

# Algoritmos

- Uma sequência finita de instruções bem definidas e não ambíguas;
- Passos para realizar uma tarefa;
- Não representa, necessariamente, um programa de computador.

# Algoritmos

- Características:

- Finitude

- Número finito de passos
  - Quantidade finita de tempo
  - Quantidade finita de recursos

- Exatidão

- Definição dos passos com clareza e precisão

- Efetividade

- Cumprir o seu propósito



IME-USP

# Algoritmos

- Algoritmos podem ser especificados de várias formas, inclusive em português.

## Exemplo 1: Fritar um ovo?

- 1 .
- 2 ..
- 3 ...
- 4 ....

## Exemplo 2: Somar dois valores?

- 1 .
- 2 ..
- 3 ...
- 4 ....

# Algoritmos

## Exemplo 3: Fazer uma busca na internet

- 1 ..
- 2 ..
- 3 ...
- 4 ....

## Exemplo 4: Converter polegadas para centímetros

- 1 ..
- 2 ..
- 3 ...
- 4 ....

# Algoritmos

## Exemplo 5: Abrir uma porta

- 1 .
- 2 ..
- 3 ...
- 4 ....

## Exemplo 6: Dada uma lista de idades de pessoas, informar a idade da pessoa mais velha

- 1 .
- 2 ..
- 3 ...
- 4 ....

# Algoritmos

Para Praticar

Escolha uma atividade da sua rotina e escreva um algoritmo para ela

# Algoritmos

Resolver os exercícios disponíveis no SAW: "**Exercícios - Elaboração de Algoritmos**"

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

- Formas de Representação

- **Linguagem natural:** escrita do algoritmo na forma de uma descrição narrativa, utilizando linguagem natural.
- **Pseudocódigo:** uma mistura da linguagem natural com a linguagem de programação para escrever os algoritmos.
- **Diagrama de blocos ou fluxograma:** representação do algoritmo graficamente, utilizando figuras geométricas.

# Algoritmos - Linguagem natural

- Linguagem natural
  - Independente de linguagem de programação.
  - Deve ser fácil de se interpretar e de codificar.
  - Deve ser o intermediário entre a linguagem falada e a linguagem de programação.

# Algoritmos - Linguagem natural

- Regras para construção:

- Usar somente um verbo por frase;
- Sempre pensar que o algoritmo será lido por pessoas que não entendem de programação;
- Usar frases curtas e simples;
- Ser objetivo;
- Evitar ambiguidade.

# Algoritmos - Linguagem natural



- Exemplo: Calcular a média aritmética de um aluno. O aluno realizará três provas: A1, A2 e A3.
  - Quais são os dados de entrada?
  - Qual será o processamento a ser utilizado?
  - Quais serão os dados de saída?

# Algoritmos - Linguagem natural

- Algoritmo:

- 1 Receba a nota da prova A1.
- 2 Receba a nota da prova A2.
- 3 Receba a nota da prova A3.
- 4 Some todas as notas e divida o resultado por 3.
- 5 Mostre o resultado da divisão.

# Algoritmos - Linguagem natural

- Identifique os dados de entrada, processamento e saída no algoritmo abaixo:
  - 1 Receba o código do produto.
  - 2 Receba o valor do produto.
  - 3 Receba a quantidade do produto.
  - 4 Calcule o valor total do produto (quantidade do produto \* valor do produto).
  - 5 Mostre o código do produto e seu valor total.

# Algoritmos - Linguagem natural

- Elabore um algoritmo para calcular e imprimir a área de um círculo.

# Algoritmos - Linguagem natural

- Elabore um algoritmo que dados 2 números e imprima o maior

# Algoritmos - Linguagem natural

Resolver os exercícios disponíveis no SAW.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Algoritmos

- Representação

- Linguagem natural;
- **Diagrama de blocos ou fluxograma;** ←
- Pseudocódigo.

# Algoritmos - Fluxograma

- É um diagrama que representa o passo a passo de um algoritmo.
- Possui um conjunto de símbolos padronizados.
- É representado pelo diagrama de blocos.
- Facilita a visualização dos passos de um processamento.

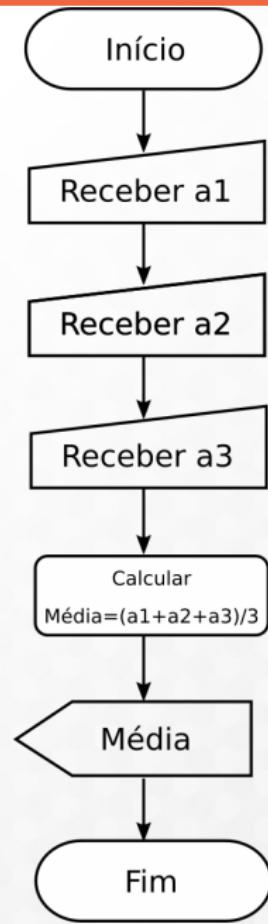
# Algoritmos - Fluxograma

Símbolo	Função
Terminal 	Indica o início e o fim de um processamento.
Processamento 	Processamento em geral. Exemplo: cálculo de dois números.
Entrada de dado manual 	Indica entrada de dados através do teclado.
Exibir 	Mostra informações ou resultados.
Decisão 	Indica uma decisão a ser tomada (desvios).

# Fluxograma: chupar bala

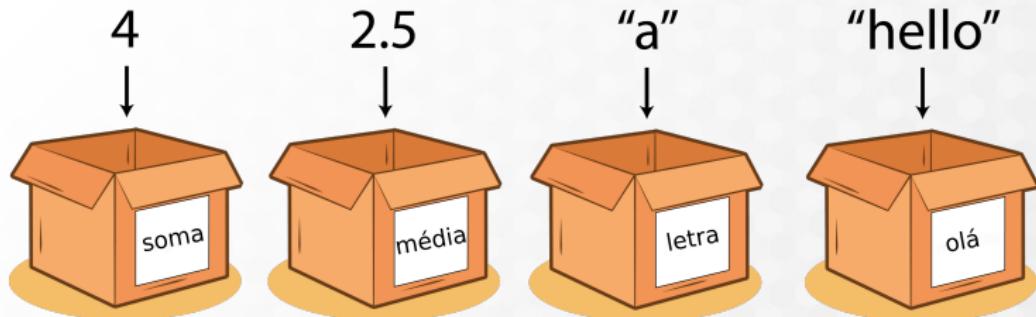


# Fluxograma: calcular média aritmética de um aluno

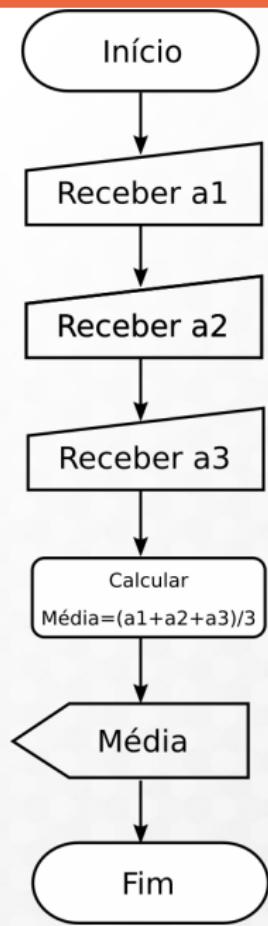


# Variáveis

- As variáveis são usadas representar valores que serão utilizados pelo algoritmo.
- Uma variável precisa de um nome para ser acessada e modificada ao longo do tempo.



# Quais são as variáveis?



# Algoritmos - Fluxograma

- Exercício 1.

- Criar o fluxograma para calcular a média de um aluno e mostrar o resultado:
  - Média  $\geq 5.0 \rightarrow$  "Aprovado";
  - Média  $< 5.0 \rightarrow$  "Reprovado".



Área para resolução

# Algoritmos - Fluxograma

- Exercício 2.

- Criar o fluxograma para informar se uma pessoa é maior ou menor de idade.

Área para resolução

# Algoritmos - Fluxograma

Resolver os exercícios disponíveis no SAW.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Algoritmos

- Representação

- Linguagem natural;
- Diagrama de blocos ou fluxograma;
- **Pseudocódigo.** ←

# Algoritmos - Pseudocódigo

- É intermediário entre a linguagem natural e uma linguagem de programação.
- Utiliza um conjunto restrito de palavras-chave.
- Não requer toda a rigidez sintática necessária numa linguagem de programação.
- Foco na lógica do algoritmos e não no formalismo de representação.

# Algoritmos - Pseudocódigo

---

## Algoritmo 1 Exemplo de Pseudocódigo.

---

```
leia ( $x, y$ ) {Esta linha é um comentário}
se  $x > y$  então
    escreva (" $x$  é maior")
senão
    se  $y > x$  então
        escreva (" $y$  é maior")
    senão
        escreva (" $x$  e  $y$  são iguais")
fim-se
fim-se
```

---

# Algoritmos - Pseudocódigo

- As palavras **leia**, **se**, **então**, **senão**, **senão-se**, **fim-se** e **escreva** são palavras-chave que representam estruturas presentes em todas as linguagens de programação.
- Não é necessário se preocupar com detalhes de sintaxe ou formatos de entrada e saída dos dados.

# Algoritmos - Pseudocódigo

- Exercício 1.
  - Criar um pseudocódigo para calcular o preço da passagem do metrô.

# Algoritmos - Pseudocódigo

- Exercício 2.
  - Criar um pseudocódigo para calcular a área de uma mesa retangular.

# Algoritmos

- Exercício 3 - Escreva os seguintes algoritmos em pseudocódigo
  - Somar dois valores.
  - Converta polegadas em centímetros.
  - Calcule a média de um aluno e mostre o resultado.
  - Informar se uma pessoa é maior ou menor de idade.

# Algoritmos - Pseudocódigo

Resolver os exercícios disponíveis no SAW.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# De algoritmos a programas

- Como transformar um algoritmo em linguagem que o computador entenda?
- Deve ser capaz de expressar tudo o que o computador pode fazer.
- Não pode ser ambígua.

# Linguagem de programação

- O que é uma linguagem?
  - Linguagem de sinais
  - Linguagem escrita

“Linguagem pode se referir tanto à capacidade especificamente humana para aquisição e utilização de sistemas complexos de comunicação, quanto à uma instância específica de um sistema de comunicação **complexo**”

# Linguagem de programação

“... é um método **padronizado** para comunicar instruções para um computador.”

- Permite especificar:
  - Os dados envolvidos;
  - Ações a serem tomadas;
  - Tratamento de diversas situações.

# Linguagem de programação

“... é um método padronizado para comunicar instruções para um computador.”

- Exemplos de linguagens de programação:
  - C
  - C++
  - Java
  - Python
  - iVProg
  - Pascal
  - PHP
  - Fortran

# Linguagem de programação

## Exemplo de código

The screenshot shows a visual programming interface with a light blue background. On the left, there's a vertical toolbar with icons for file operations, a search bar, and a help section. The main area contains a flowchart for a function named 'inicio'. The code starts with initializing variables 'a' and 'b' to 0. It then reads two integers from the user. A decision block checks if 'a' is greater than 'b'. If true, it prints 'a'; otherwise, it prints 'b'. The flowchart uses colored blocks: pink for variable assignment, yellow for input and output, and grey for conditionals and loops.

```
funcao vazio inicio ()  
    inteiro a ← 0  
    inteiro b ← 0  
    leia (a)  
    leia (b)  
    se (a > b) I  
        escreva (a)  
    senao  
        escreva (b)
```

Figura: iVProg

```
1 #include <stdio.h>  
2 int main(){  
3     int a = 0;  
4     int b = 0;  
5     scanf("%d %d", &a, &b);  
6     if (a>b){  
7         printf("%d", a);  
8     } else{  
9         printf("%d", b);  
10    }  
11 }
```

Figura: C



IME-USP

# Linguagem Interpretada X Compilada

- Tradução do código fonte para código de máquina.
- **Interpretada:**
  - O programa é executado conforme vai sendo traduzido.
- **Compilada:**
  - Todo o código fonte é traduzido antes de ser executado.

# Linguagem Interpretada X Compilada

Source code:

hello.c



→ COMPILER →

Machine code:

11010  
11011  
10001

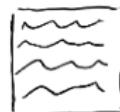
Program (also  
called binary,  
executable ...)

run the  
program → result



Source code:

hello.py



→ INTERPRETER → result



# Paradigma

- Estrutural
  - Sequência, decisão e iteração
  - Estruturas, como sub-rotinas e funções
  - Resolver problemas mais simples e diretos
  - C, Cobol, Pascal e Perl
- Orientada a Objetos
  - Modelar o mundo real no domínio do problema
  - Java, C++

- Programação visual
  - Mais próxima da linguagem natural
  - Código feito em blocos
  - Uso mais simplificado
  - Scratch, EV3 (Lego), iVProg

# Alto x Baixo nível

- Baixo nível

- Linguagem de máquina: **0s e 1s**
- Assembly

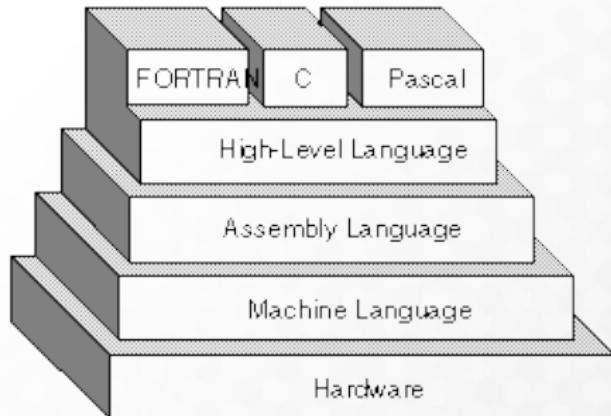
LOAD	NOTA
ADD	NOTAEXTRA
STORE	NOTAFINAL

- Alto nível

- Mais distantes da máquina e mais próximas de linguagens naturais (inglês, português, etc.).
- Mesmo mais compreensíveis, elas não são ambíguas.
- Um compilador as transforma em código executável.

NOTAFINAL = NOTA + NOTAEXTRA

# Alto x Baixo nível



# Dúvidas?

# Introdução à Programação

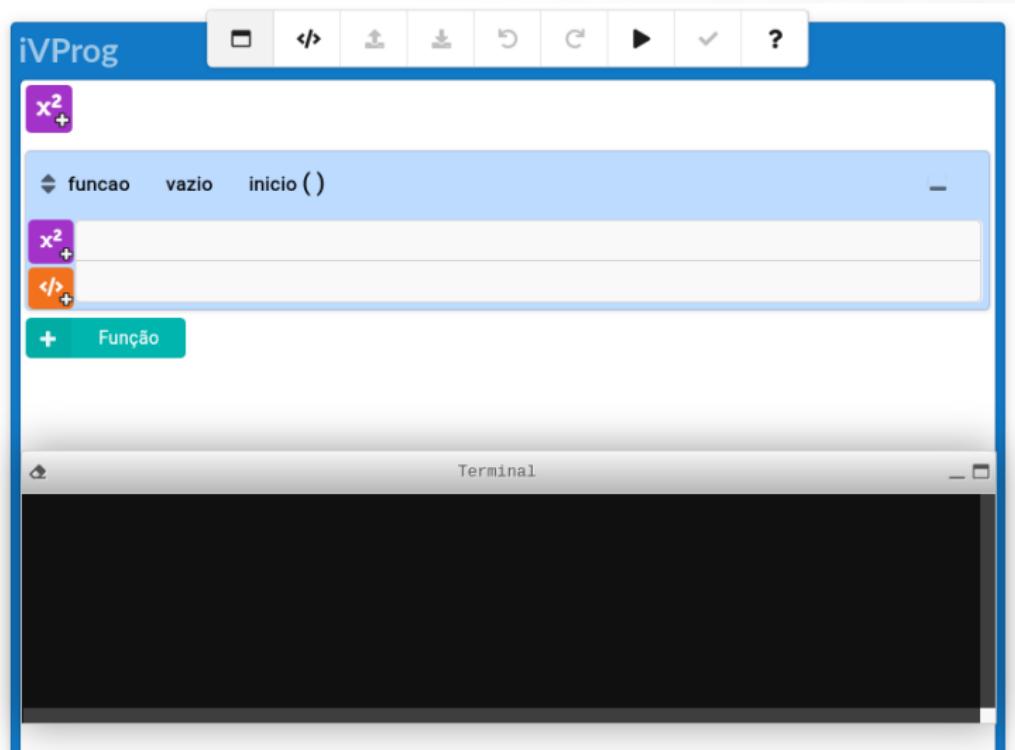
Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

- Ferramenta de programação visual.
- Permite focar no raciocínio lógico e algorítmico.
- Podemos utilizá-lo para elaborar, executar, testar e avaliar algoritmos.

# iVProg



- Criação de variáveis.
- Comando de entrada de dados.
- Comando de saída de dados.
- Comando de atribuição de valores.

- Variáveis



Botão para criação de variáveis

inteiro

variavel\_0

$\leftarrow$  0

inteiro

variavel\_1

$\leftarrow$  0

Elementos de uma variável

inteiro

variavel\_0

$\leftarrow$  0

tipo

nome

conteúdo

- Tipos de variáveis

- Numéricos

- **Inteiro:** número sem componente fracional. Exemplo: idade de uma pessoa, número da casa na rua.
- **Real:** valores com casas decimais. Exemplo: preço da gasolina, cotação do dólar.

- Textuais

- **Caractere:** representa um único caractere. Exemplo: 'a', '@'.
- **Texto:** representa uma cadeia de caracteres. Exemplo: nome de uma pessoa, endereço.

- **Lógico:** verdadeiro ou falso. Exemplo: resultado de uma comparação.



IME-USP

leia ( variavel\_0 ▾ )



- Entrada/leitura de dados

- Solicita ao usuário para fornecer algum dado pelo teclado.
- Armazena o valor informado na variável associada ao comando.
- O valor antigo da variável é perdido.

↑ escreva ( variavel\_0 )  

✗

- Saída/escrita de dados

- Imprime o conteúdo informado na instrução:

- Variável
  - Texto
  - Expressão aritmética
  - Expressão lógica

```
← variavel_0 ▾    recebe    variavel_1 * 3  🔒  ✕
```

## ● Atribuição

- Altera o conteúdo da variável recebedora, substituindo-o pelo resultado da expressão.

Resolver os exercícios disponíveis no SAW.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

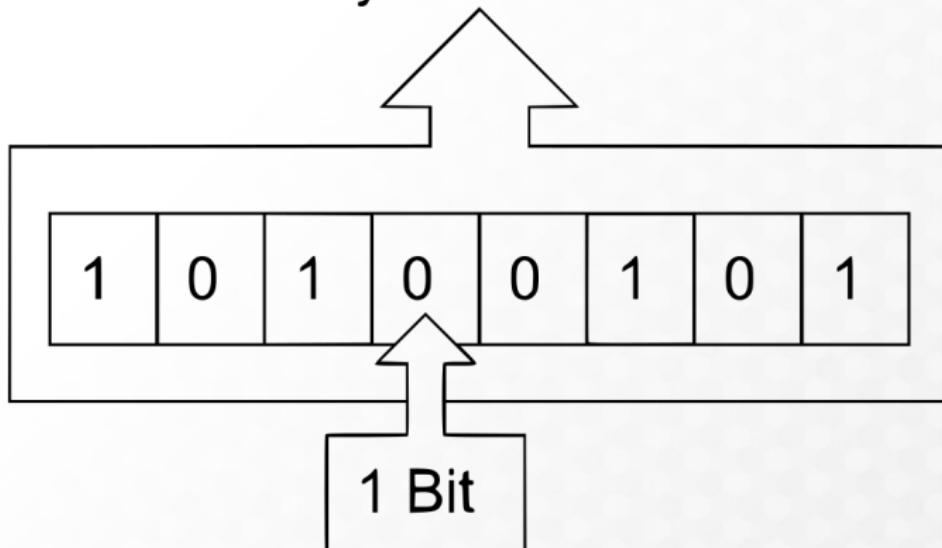
Verão 2023

# bits e bytes

- Os computadores entendem impulsos elétricos, representados por 1 ou 0 (presença ou ausência de eletricidade)
- Cada impulso elétrico é chamado de bit (**BInary digiT**)
- Um conjunto de 8 bits reunidos forma um **byte**

bits e bytes

1 Byte = 8 Bits



Adaptado de Frank Carmody, 2016

- **byte:** consegue qualquer caractere possível (letras, números, sinais de pontuação...)
- Logo, para representar a letra 'a', é necessário 1 byte, ou seja, 8 bits.
- Se um byte tem 8 bits, logo, existem 256 combinações possíveis de bytes:  $2^8$
- Dois bytes, ou 16 bits, podem ter 65.535 combinações diferentes.

# bits e bytes

- Exemplo: para armazenar "BRUNA" na memória do computador, precisamos de 5 bytes, ou seja, um para cada letra ou caractere.
- Tabela ASCII, combina números binários com símbolos.
- "BRUNA":  
01000010 01010010 01010101 01001110 01000001
- Conversor de texto para binário:  
<https://cryptii.com/pipes/text-to-binary>

# Decimal - Binary - Octal - Hex – ASCII Conversion Chart

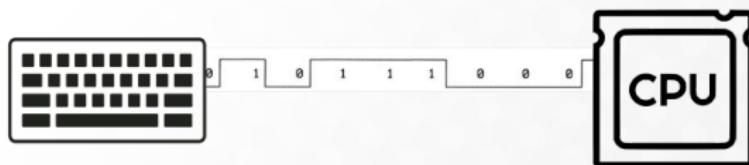
Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	'
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	\$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	000001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	000001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	000001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	000001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	000001000	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	000001001	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	000001010	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	000001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	000100000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	000100001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	000100010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	000100111	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	000101000	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	000101001	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	000101100	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	000101111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	000110000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	000110001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	000110010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	000110011	033	1B	ESC	59	00111011	073	3B	:	91	01011011	133	5B	[	123	01111011	173	7B	{
28	000110100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	
29	000110101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	000111100	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	000111111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

# Memória do computador

Endereço	Conteúdo
0	00000011
1	00000001
2	00000101
3	00000100
4	00000011
5	00000000
6	00000000
7	00000000
8	00000000



# Envio de dados



## Capacidades de armazenamento

- 1 Byte = 8 bits
- 1 kilobyte (KB ou Kbytes) = 1024 bytes
- 1 megabyte (MB ou Mbytes) = 1024 kilobytes
- 1 gigabyte (GB ou Gbytes) = 1024 megabytes
- 1 terabyte (TB ou Tbytes) = 1024 gigabytes
- 1 petabyte (PB ou Pbytes) = 1024 terabytes
- 1 exabyte (EB ou Ebytes) = 1024 petabytes
- 1 zettabyte (ZB ou Zbytes) = 1024 exabytes
- 1 yottabyte (YB ou Ybytes) = 1024 zettabytes

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

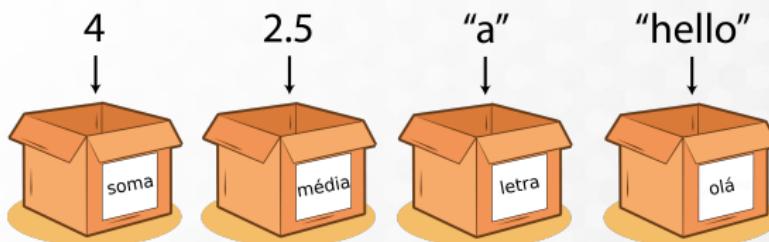
Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Variáveis

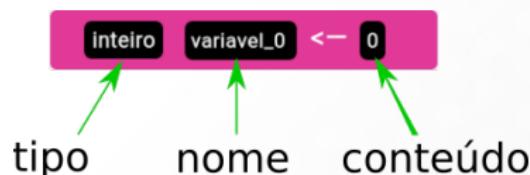
## Definição

Variáveis são locais onde armazenamos valores na memória. Toda variável é caracterizada por um **nome**, que a identifica em um programa, e por um **tipo**, que determina o que pode ser armazenado naquela variável.



# Declarando uma variável

em iVProg:



em C:  
Declaração:

```
int variavel_0;
```

A diagram illustrating a variable declaration in C. The code "int variavel\_0;" is shown. Two green arrows point from the words "tipo" and "nome" below to the "int" and "variavel\_0" tokens respectively in the code.

Declaração e **inicialização**:

```
int variavel_0 = 0;
```

A diagram illustrating a variable declaration and initialization in C. The code "int variavel\_0 = 0;" is shown. A green arrow points from the word "operador de atribuição" to the assignment operator (=). Another green arrow points from the word "valor inicial" to the value "0".

- Nome que identifica um endereço de memória.
- Linguagem C é sensível a maiúsculas e minúsculas.
- Para nomes de variáveis, utilize os caracteres:
  - "A-Z"
  - "a-z"
  - "0-9"
  - e *underline* "\_".

## Atenção!

**Identificadores** não podem  
começar por números!  
E não use caracteres especiais  
(á, é, ç, ã...).

# Variáveis - Identificadores

- Convenções (padrões de nomenclatura):
  - lower\_case\_with\_underscores
  - mixedCase
- Exemplos:
  - Identificadores válidos:
    - precoUnitario, qtdeEstoque, idade, autor, dataNasc.
    - preco\_unitario, qtde\_estoque, idade, autor, data\_nasc.
  - Identificadores inválidos:
    - preço, 5x, idade cliente, \$valor, id@de.



# Palavras reservadas

- Palavras reservadas, também conhecidas como palavras-chave, não podem ser utilizadas com identificadores.
- São reservadas pela linguagem para representar certas características, por exemplo, tipos de variáveis.

# Palavras reservadas iVProg

- Dependem da linguagem utilizada.
- Para o português são:

inteiro	real	logico
caractere	cadeia	vazio
se	senao	repita
repita_para	repita_enquanto	escolha
caso	pare	devolva
funcao	vazio	const



# Palavras reservadas C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

# Tipos básicos de dados

iVProg

- inteiro
- real
- caractere
- cadeia
- logico

C

- int
- float
- double
- char

# Tipo inteiro/int

- Palavra chave: inteiro (iVProg), int (C)
- Representa números inteiros positivos e negativos
- No C:
  - 32 bits armazenáveis (4 bytes), 1 bit para sinal
  - Pode armazenar valores de -2.147.483.648 a 2.147.483.647

# Tipo real/float

- Palavra chave: real (iVProg), float (C)
- Representa números fracionários positivos e negativos
- No C:
  - **Precisão simples!**
  - 32 bits armazenáveis (4 bytes)

# Tipo double

- Palavra chave: double (C)
- Representa números fracionários positivos e negativos
  - **Precisão dupla!**
  - 64 bits armazenáveis (8 bytes)
  - Não existe no iVProg

# Tipo caractere/char

- Utilizado com códigos de caractere
  - Tabela ASCII (*American Standard Code for Information Interchange*), UTF8
- No C:
  - É na verdade um número inteiro de 8 bits
  - 8 bits armazenáveis (1 byte)
  - Caractere representado deve sempre estar entre aspas simples
  - char letra = 'a'

# Tipo caractere/char - Tabela ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	-	127	7F	[DEL]

# Tipo caractere/char - Tabela ASCII Extendida

0	32	64	¤	96	`	128	¢	160	à	192	Ł	224	ó
1	¤	33	!	65	À	97	à	129	ü	161	í	193	ł
2	¤	34	"	66	฿	98	฿	130	é	162	ó	194	™
3	♥	35	#	67	₵	99	₵	131	â	163	ú	195	ƒ
4	♦	36	฿	68	฿	100	฿	132	ä	164	ñ	196	-
5	♣	37	₩	69	£	101	£	133	à	165	ñ	197	†
6	♠	38	&	70	₱	102	₱	134	å	166	¤	198	ã
7	•	39	'	71	₲	103	₲	135	¤	167	¤	199	ܵ
8	■	40	(	72	₪	104	₪	136	ê	168	ܶ	200	₪
9	◊	41	)	73	₭	105	₭	137	ë	169	₭	201	₭
10	₵	42	*	74	₭	106	₭	138	è	170	߱	202	߱
11	฿	43	+	75	₭	107	₭	139	ି	171	ି	203	ି
12	♀	44	,	76	₭	108	₭	140	ି	172	ି	204	ି
13	₱	45	_	77	₭	109	₭	141	ି	173	ି	205	=
14	߱	46	.	78	߱	110	߱	142	߱	174	߱	206	߱
15	߱	47	/	79	߱	111	߱	143	߱	175	߱	207	߱
16	▶	48	߱	80	߱	112	߱	144	߱	176	߱	208	߱
17	◀	49	߱	81	߱	113	߱	145	߱	177	߱	209	߱
18	߱	50	߱	82	߱	114	߱	146	߱	178	߱	210	߱
19	߱	51	߱	83	߱	115	߱	147	߱	179	߱	211	߱
20	߱	52	߱	84	߱	116	߱	148	߱	180	߱	212	߱
21	߱	53	߱	85	߱	117	߱	149	߱	181	߱	213	߱
22	߱	54	߱	86	߱	118	߱	150	߱	182	߱	214	߱
23	߱	55	߱	87	߱	119	߱	151	߱	183	߱	215	߱
24	↑	56	߱	88	߱	120	߱	152	߱	184	߱	216	߱
25	↓	57	߱	89	߱	121	߱	153	߱	185	߱	217	߱
26	→	58	:	90	߱	122	߱	154	߱	186	߱	218	߱
27	←	59	:	91	߱	123	߱	155	߱	187	߱	219	߱
28	↔	60	<	92	߱	124	߱	156	߱	188	߱	220	߱
29	↔	61	=	93	߱	125	߱	157	߱	189	߱	221	߱
30	▲	62	>	94	߱	126	߱	158	߱	190	߱	222	߱
31	▼	63	?	95	߱	127	߱	159	߱	191	߱	223	߱



# Tipo cadeia

- Palavra chave: cadeia (iVProg)
- Representa uma cadeia de caracteres
  - Junção de 2 ou mais caracteres
- Não existe no C

# Tipo lógico

- Palavra chave: logico (iVProg)
- Representa um valor Verdadeiro ou Falso
- Em geral resultado de uma expressão lógica
  - $5 > 6$
- Não existe no C

Resolver os exercícios disponíveis no SAW.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Entrada de dados

- Leitura de informação enviada a partir do teclado.
- Função **scanf** da **stdio.h**.
  - Parâmetros da função:
    - 1 Um texto, informando os tipos e formatos das variáveis que serão lidas.
    - 2 Uma ou mais variáveis separadas por vírgula.
  - Aguarda o usuário digitar um valor.
  - Atribui o valor recebido à variável indicada.

# Entrada de dados

- Exemplo de uso da função `scanf`:

```
#include <stdio.h>

int main () {
    int idade;
    printf("Informe sua idade: ");
    scanf("%d", &idade);
    printf("A idade informada foi %d.\n", idade);
    return 0;
}
```

# Entrada de dados

- Leitura de várias variáveis com `scanf`:

```
#include <stdio.h>

int main () {
    int x, y, z;
    printf("Digite três inteiros: ");
    scanf("%d %d %d", &x, &y, &z);
    // Como imprimir os três valores lidos?
    return 0;
}
```

# Operador de endereço

- O operador `&` retorna o endereço de uma determinada variável.
- Exemplo:

```
#include <stdio.h>

int main () {
    int idade = 22;
    printf ("%p", &idade);
    return 0;
}

Saída:
$ 0x7fff8ad4f274
```



# Operador de endereço

- Usa-se o & para indicar que o valor recebido do teclado deverá ser colocado no endereço referente a uma variável.
- Esquecer de colocar o & no scanf é comum e causará erros de execução.



# Formatadores de entrada

Código	Tipo de dado
%c	caractere
%s	série de caracteres
%d	inteiro
%u	inteiro sem sinal
%l	inteiro longo
%f	número em ponto flutuante
%lf	double
%p	endereço de memória

# Saída de dados

- Podemos imprimir texto puro e o conteúdo de variáveis. **E também misturá-los!**
- Função **printf** da **stdio.h**.
- Parâmetros da função:
  - Um texto, podendo incluir um símbolo no meio, representando que o trecho deve ser substituído por uma variável.
  - Se o texto possui algum símbolo de variável, então, a lista de variáveis, separadas por vírgula.

```
printf("O saldo da conta %d é R$ %f", numero_conta,  
      saldo_atual);
```



# Saída de dados

```
printf("O saldo da conta %d é R$ %f", numero_conta,  
      saldo_atual);
```

- Nesse exemplo, %d deve ser substituído por uma variável do tipo inteiro e %f deve ser substituído por uma variável do tipo ponto flutuante.

# Saída de dados

- Formatadores para números inteiros

Formatador	Resultado impresso
<code>printf("%4d", soma);</code>	<espaço><espaço>10
<code>printf("%04d", soma);</code>	0010
<code>printf("%6.04d", soma);</code>	<espaço><espaço>0010

- Números grandes:

```
printf ("%d", 4000000000);
```

Saída: -294967296

```
printf ("%u", 4000000000);
```

Saída: 4000000000

# Saída de dados

- Formatadores para ponto flutuante

- **%f**

- Escreve um ponto flutuante na tela, sem formatação do conteúdo.
- Exemplo:

```
printf("%f", 16.5);  
Saída: 16.500000
```

- **%e**

- Escreve um ponto flutuante na tela, em notação científica.
- Exemplo:

```
printf("%e", 18.06816);  
Saída: 1.806816e+01
```



IME-USP

# Saída de dados

- Formatadores para ponto flutuante
- **%.2f**
  - Escreve um ponto flutuante na tela, com duas casas decimais.
  - Exemplo:

```
printf("%.2f", 22.0);  
Saída: 22.00
```

- **%6.2f**
  - Escreve um ponto flutuante na tela, com tamanho seis e duas casas decimais.
  - Exemplo:

```
printf("%6.2f", 22.5);  
Saída: <espaço>22.50
```



IME-USP

# Saída de dados

- Formatadores para caractere
- %c
  - Escreve um caractere na tela.
  - Exemplo:

```
printf("%c", letra);  
Saída: B
```

# Saída de dados

- Formatadores para texto (string)

- %s

- Escreve uma string na tela.
- Exemplo:

```
printf("%s", "Seja bem vindo!");  
Saída: Seja bem vindo!
```

- C não tem o tipo String

- Mais a frente veremos como guardar textos



IME-USP

# Sequências de escape

- Combinação de caracteres, iniciando com uma barra invertida: \
- É útil para imprimir caracteres de controle.

Sequência de escape	Descrição
\'	Mostra aspas simples na tela (apóstrofo)
\\"	Mostra aspas duplas na tela
\?	Mostra o ponto de interrogação na tela
\\'\	Mostra a barra na tela
\a	Alert/campainha
\b	Backspace: move o cursor uma posição para trás da linha
\n	Nova linha: move o cursor para o início da linha seguinte
\r	Carriage return: move o cursor para o início da linha atual
\t	Tabulação horizontal
\v	Tabulação vertical



# Entrada e saída de dados

Resolver os exercícios disponíveis no SAW.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Atribuição

- Atribui o valor de uma expressão a uma variável.
- A expressão será calculada e o valor armazenado em uma determinada variável.

## Atribuição no iVProg

```
← soma ▾    recebe   a + b  🔒  ✕
```

## Atribuição em C

```
soma = a + b;
```

# Atribuição

## Atribuição no iVProg



## Atribuição em C

```
soma = a + b;
```

à esquerda do operador de atribuição temos somente o nome de uma variável

à direita do operador de atribuição devemos especificar uma expressão



IME-USP

# Exemplos de atribuição

- A atribuição pode ser utilizada para todos os tipos de dados.

```
int x;
int y;
float saldo;
float preco;
float pi;
char bloco;

x = 3;
y = 4698 / 898 + 41;

bloco = 'A';

saldo = 982.66;
preco = 17.4;
pi = 3.14159;
```



# Expressões simples

- **Constante:** expressão com um valor fixo.

Exemplo: `idade = 30;`

- **Variável:** uma variável é uma expressão!

Exemplo: `nota = prova1;`

- **Endereço de uma variável:** também é uma expressão.

Exemplo: `endereco = &y;`

# Expressão

- As constantes, as variáveis e endereços de memória são expressões.
- Uma expressão pode ser o conjunto de operações aritméticas, lógicas ou relacionais.
- Exemplo: nota1 + nota2 — calcula a soma de nota1 e nota2.

## Expressões

- < expressao > - < expressao >  
Calcula a subtração de duas expressões.
- < expressao > + < expressao >  
Calcula a soma de duas expressões.
- < expressao > \* < expressao >  
Calcula o produto de duas expressões.
- < expressao > / < expressao >  
Calcula o quociente de duas expressões.
- < expressao > % < expressao >  
Calcula o resto da divisão (inteira) de duas expressões.
- - < expressao >  
Inverte o sinal da expressão.

# Expressões

- Qual o valor das expressões abaixo?

- $6 - 4 \% 2:$
- $8 * 10 \% 3:$

# Expressões

- Qual o valor das expressões abaixo?

- $6 - 4 \% 2 : 6$
- $8 * 10 \% 3 : 2$

# Precedência de operadores

- Ordem em que os operadores serão calculados.
  - 1 \* e / → na ordem que aparecerem na expressão;
  - 2 %
  - 3 + e - → na ordem que aparecerem na expressão.

# Como modificar a precedência?

- Parênteses!

- ( < expressao > )
  - Também é uma expressão.
  - Será resolvida primeiro para que então outras expressões sejam resolvidas.
  - Precisamos utilizar os parênteses quando a ordem de precedência dos operadores não atende às nossas necessidades.
  - Não existe um limite de uso dos parênteses em uma expressão.
  - **Atenção!** Ao abrir um parêntese, não se esqueça de fechá-lo!

# Como modificar a precedência?

- $6 - 4 \% 2: 6$
- $(6 - 4) \% 2:$
- $8 * 10 \% 3: 2$
- $8 * (10 \% 3):$



# Como modificar a precedência?

- $6 - 4 \% 2: 6$
- $(6 - 4) \% 2: 0$
- $8 * 10 \% 3: 2$
- $8 * (10 \% 3): 8$

# Incremento e decremento

- Incremento (++)
- Decremento (--)
- Funcionam como expressões.
- Incrementam ou decrementam o valor de uma variável em uma unidade.
- Exemplos:

$a++ \rightarrow a = a + 1$

$f-- \rightarrow f = f - 1$

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

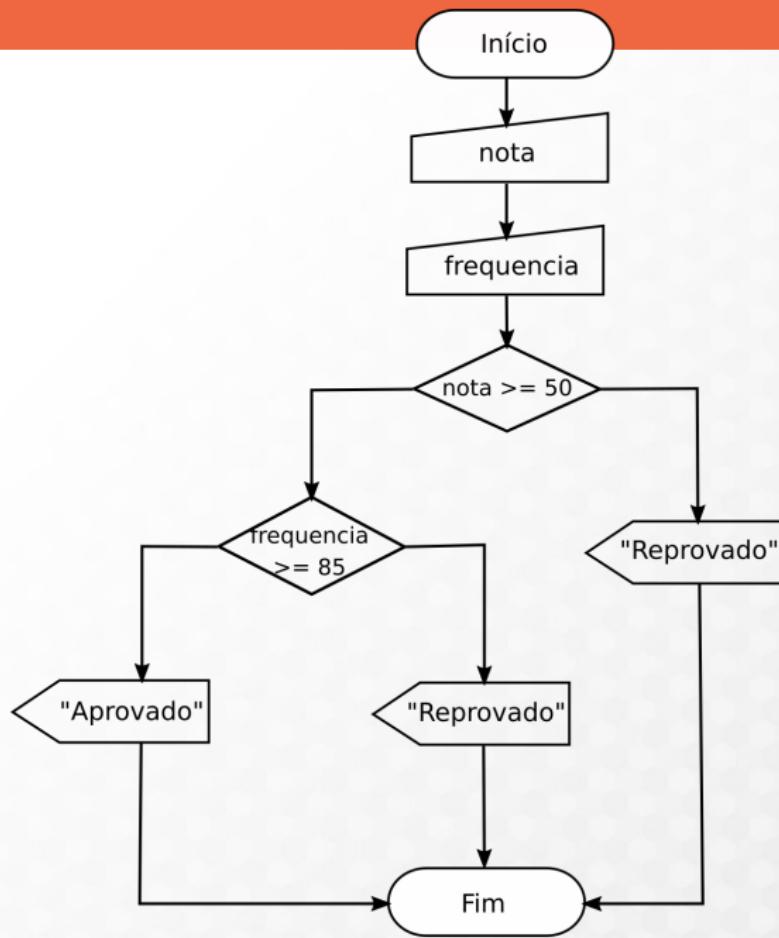
Verão 2023

# Problema

Precisamos criar um programa que receba a nota de um aluno e decida se o mesmo foi **aprovado ou reprovado**.

Critérios para aprovação: **nota mínima de 50.00 e presença mínima de 85%**.

# Solução



# Operadores relacionais

- São utilizados para comparar valores.
- O resultado da comparação é um valor booleano (verdadeiro ou falso).
- Exemplos:
  - Nota é no mínimo 50?
  - Presença é no mínimo 85%?

# Operadores relacionais

Operador	Descrição	Exemplos
<code>==</code>	igual a	$2 == 2$ $x == 2$
<code>!=</code>	diferente de	$3 != 4$ <code>area != 4</code>
<code>&gt;</code>	maior que	$4 > 3$ <code>idade &gt; 18</code>
<code>&gt;=</code>	maior que ou igual a	$4 >= 3$ <code>primo &gt;= 3</code>
<code>&lt;</code>	menor que	$3 < 9$ $x < y$
<code>&lt;=</code>	menor que ou igual a	$9 <= 9$ <code>soma &lt;= 100</code>

# Operadores relacionais

- Exemplos:

$5 == 5$	
$5.6 != 5.60$	
$3.001 > 3$	
$1265 \geq 3$	
$-1 < -8$	
$'A' == 'a'$	



# Operadores relacionais

- Exemplos:

$5 == 5$	Verdadeiro
$5.6 != 5.60$	Falso
$3.001 > 3$	Verdadeiro
$1265 >= 3$	Verdadeiro
$-1 < -8$	Falso
$'A' == 'a'$	Falso



# Operadores relacionais - exemplos

- Quais as expressões para as situações abaixo?
  - Nota é no mínimo 50?
  - Presença é no mínimo 85%?
  - Tenho dinheiro suficiente para pagar uma conta de R\$100.00?
  - 2 pessoas pagaram o mesmo valor em uma divisão de conta?

# Operadores Lógicos

- Operam valores lógicos
- *expressão\_lógica operador expressão\_lógica*
- O resultado é uma expressão lógica

Operador	Descrição	Exemplo
<code>&amp;&amp;</code>	E	<code>a &amp;&amp; b</code>
<code>  </code>	OU	<code>a    b</code>

# Operadores Lógicos - Tabela verdade

A	B	$A \&\& B$	$A    B$
F	F	F	F
F	V	F	V
V	F	F	V
V	V	V	V

# Operadores lógicos

- Suponha que nota\_aluno e presenca\_aluno são variáveis do tipo float.
- Como escrever as condições de aprovação?

A nota do aluno é no mínimo 50.00

E

a presença do aluno é no mínimo 85%?

`nota >= 50.0 && frequencia >= 85`

# Precedência entre operadores

( )
!
* /
%
+ -
<= >= > <
== !=
&&



- Não separe os símbolos dos operadores relacionais ( $!=$ ,  $==$ ,  $>=$ ,  $<=$ ).
  - Certo: (idade  $\geq$  18)
  - Errado: (idade  $> = 18$ )
- Não inverta a ordem dos símbolos dos operadores relacionais ( $!=$ ,  $==$ ,  $>=$ ,  $<=$ ).
  - Certo: (idade  $\geq$  18)
  - Errado: (idade  $=>18$ )
- Não confunda o operador de comparação  $==$  com o operador de atribuição  $=$

- Até agora, vimos uma estrutura de código: sequencial.
- E se quisermos que o código tenha um fluxo alternativo?
- Se quisermos introduzir uma condição?

## Estrutura Condicional

# Estrutura de seleção

- Utilizada para alterar o fluxo do programa.
- Conceito:

```
se (condição) então:  
    (bloco de instruções 1)  
senão:  
    (bloco de instruções 2)  
fim se
```

# Estrutura de seleção - iVProg

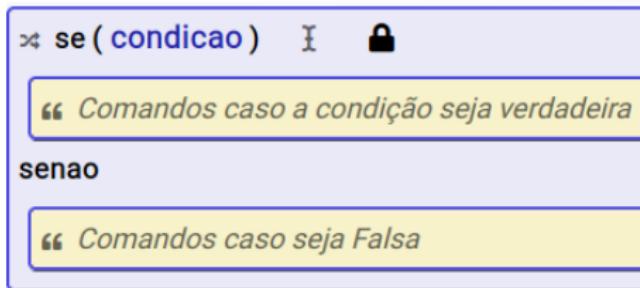


Figura: Se, Senão no iVProg

Exemplo: imprimir na tela a mensagem "Aprovado" caso o aluno tenha obtido média maior ou igual a 50.0 ou a mensagem "Reprovado", caso contrário.

# Estrutura de seleção - C

Na linguagem C:

```
if (condição) {  
    /* comandos caso condição seja verdadeira */  
} else {  
    /* comandos caso condição seja falsa */  
}
```

Exemplo: imprimir na tela a mensagem "Aprovado" caso o aluno tenha obtido média maior ou igual a 50.0 ou a mensagem "Reprovado", caso contrário.

# Estrutura de seleção - C

Se dentro do *if* você colocar apenas uma instrução, o uso das chaves **{ }** é opcional.

Exemplo:

```
if (media >= 50.0) {  
    printf("Aprovado!");  
}
```

# Estrutura de seleção - C

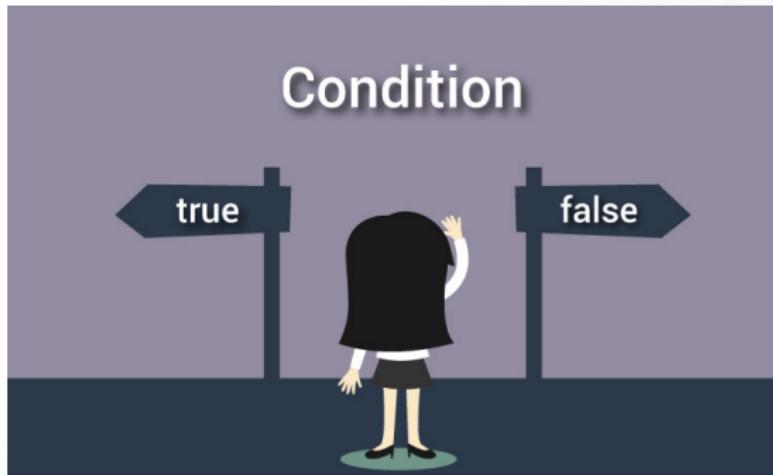
Se dentro do *if* você colocar apenas uma instrução, o uso das chaves `{ }` é opcional.

Exemplo:

```
if (media >= 50.0)
    printf("Aprovado!");
```

Mesmo código:

```
if (media >= 50.0) printf("Aprovado!");
```



- Elaborar o algoritmo de aprovação/reprovação no iVProg e em C.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Operador ternário

# Operador ternário

- Estrutura de seleção.
- Sintaxe:
  - `a ? b : c`
- `<expressaoLogica> ? <se verdadeira> : <se falsa>`
- Estrutura de seleção com **retorno de valor**.

# Operador ternário

- Exemplo 1:
  - Criar um programa que leia dois números e informe qual o maior deles.

# Operador ternário

- Exemplo 1:

- Solução:

## Atribuição condicionada:

```
if (num1 >= num2) {  
    maior = num1;  
} else {  
    maior = num2;  
}
```

- Como podemos fazer:

```
maior = num1 >= num2 ? num1 : num2;
```

# Operador ternário

- Exemplo 2:
  - Ler um número e exibir uma mensagem na tela informando se o número é **par** ou **ímpar**.

# Operador ternário

- Exemplo 1:

- Solução:

**Como fazemos:**

```
if (num % 2 == 0){  
    printf("O numero é par\n");  
} else {  
    printf("O numero é impar\n");  
}
```

- Como podemos fazer:

```
printf("O numero é %s\n", (num % 2 == 0 ? "par" : "impar"));
```

# Operador ternário

- Exemplo 2:
  - Ler um número e exibir uma mensagem na tela informando se o número é **par** ou **ímpar**.

# Estruturas de seleção - switch

# Estruturas de seleção - switch

- Exemplo 1:

- Criar um programa para ler dois inteiros num1 e num2. Após isto, leia um operador aritmético e calcule conforme escolhido:
  - num1 + num2
  - num1 - num2
  - num1 \* num2
  - num1 / num2

# Estruturas de seleção - switch

- Exemplo 1:

- Solução:

```
if (operacao == '+') {
    printf("%d", num1 + num2);
} else {
    if (operacao == '-') {
        printf("%d", num1 - num2);
    } else {
        if (operacao == '*') {
            printf("%d", num1 * num2);
        } else {
            printf("%d", num1 / num2);
        }
    }
}
```

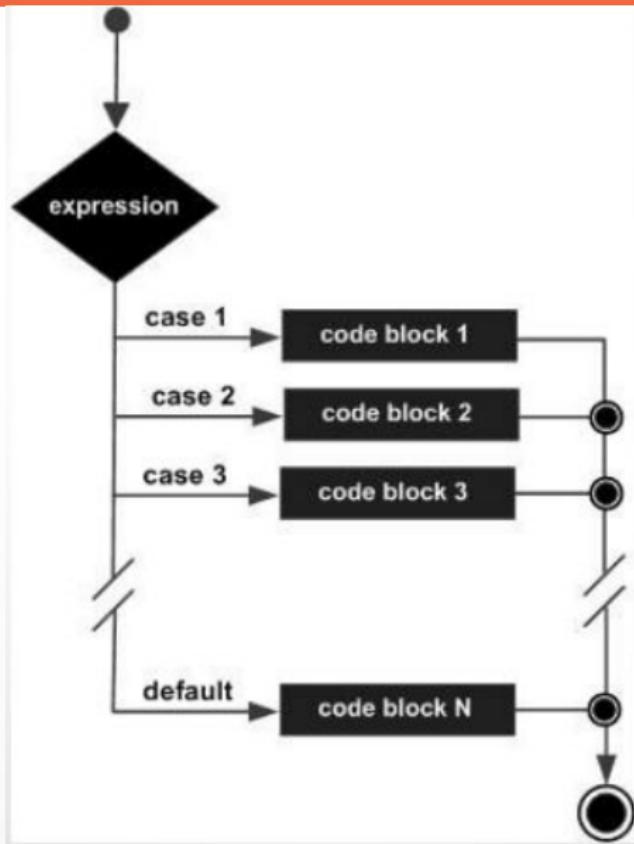
# Estruturas de seleção - switch

- Exemplo 1:
  - Como podemos fazer usando **switch**:

```
switch (operacao) {  
    case '+':  
        printf("%d", num1 + num2);  
        break;  
    case '-':  
        printf("%d", num1 - num2);  
        break;  
    case '*':  
        printf("%d", num1 * num2);  
        break;  
    case '/':  
        printf("%d", num1 / num2);  
        break;  
}
```



# Estruturas de seleção - switch



# Estruturas de seleção - switch

- Exemplo 2:

- Criar um programa para ler o conceito obtido por um aluno de uma disciplina.
  - Para conceito **A**, exibir “**Excelente!**”.
  - Para conceitos **B** e **C**, exibir “**Muito bem!**”.
  - Para conceito **D**, exibir “**É, passou!**”.
  - Para conceito **R**, exibir “**Ano que vem tem de novo!**”.
  - Caso padrão, exibir “**Conceito inválido!**”.

# Estruturas de seleção - switch

- Exemplo 2:

```
switch (conceito) {  
    case 'A':  
        printf("Excelente!");  
        break;  
    case 'B':  
    case 'C':  
        printf("Muito bem!");  
        break;  
    case 'D':  
        printf("É, passou!");  
        break;  
    case 'R':  
        printf("Ano que vem tem de novo!");  
        break;  
    default:  
        printf("Conceito inválido!");  
}
```

O que ocorre nos casos que não utilizamos break?

# Estruturas de seleção - switch

- Suporta apenas igualdade.
- Atenção ao uso do **break**.
- Os casos devem ser números inteiros.
  - Caracteres são interpretados como inteiros pela tabela ASCII
  - case 'A' é o mesmo de case 65
  - case 'b' é o mesmo de case 98

# Estruturas de seleção - switch

- Faça um teste!
- Crie um programa que lista um menu de opções para o usuário.
- Trate a opção escolhida pelo usuário (leia os dados e faça a operação).

BANCO MM

1 - Ver o saldo

2 - Fazer um saque

3 - Fazer uma transferência

4 - Imprimir o extrato

Escolha sua opção:



# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Função (Sub-rotina)

- É um conjunto de códigos auto-contidos que visam cumprir uma tarefa específica.
- Normalmente, uma função “recebe” dados, os processa e “retorna” um resultado.
- Uma vez definida, uma função pode ser utilizada quantas vezes forem necessárias sem precisar reimplementá-la.
- Permitem abstrair um dado processo, não sendo necessário conhecer como foi implementada.

# Função (Sub-rotina)

- Já utilizamos algumas funções até agora:
  - **printf**
  - **scanf**
  - **main**
- A maioria delas “recebe” dados através de um mecanismo que chamamos de **parâmetro**.

# Função (Sub-rotina)

- Os parâmetros de uma função possuem dois comportamentos distintos:
  - Passagem por cópia - o valor da variável é copiado para “dentro” da função.
  - Passagem por referência - uma referência (endereço) da variável é passado para “dentro” da função.
- Ex: **printf** tem parâmetros por cópia enquanto **scanf** tem por referência.
- Iremos trabalhar primeiramente com os parâmetros de passagem por cópia.

# Função (Sub-rotina)

- **Atenção!** Evite leituras e impressões dentro das funções.
- Sintaxe:

```
tipo_retorno identificador (tipo param1,...) {  
    comandos...  
}
```

- Note que esta sintaxe dos parâmetros é para passagem por cópia!
- Ex:

```
int soma (int a, int b) {  
    return a + b;  
}
```

# Função (Sub-rotina)

*Nome da função*

*Tipo de retorno*

*Parâmetros*

*Cabeçalho da função*

*Corpo da função*

```
int soma(int x, int y) {  
    int total;  
    total = x + y;  
    return total; ← Comando de retorno  
}
```

# Chamada de função

- Executa os comandos definidos no corpo da função.

Variável que  
receberá o  
resultado da  
função

↓

Nome da  
função Parâmetros

↓

resultado = soma(15, a);

# Chamada de função

```
#include <stdio.h>
```

```
int soma(int x, int y) {  
    int total;  
    total = x + y;  
    return total;
```

```
}
```

resultado  
= total

```
int main() {  
    int resultado;  
    int a, b;
```

```
scanf("%d %d", &a, &b);
```

```
resultado = soma(a, b);
```

```
return 0;
```

```
}
```

x = a  
y = b



# Exercício

- Quais as saídas do seguinte código?

```
#include <stdio.h>

int dobro (int a) {
    a = a * 2;
    return a;
}

int main () {
    int a = 5;
    printf ("%d\n", a);
    printf ("%d\n", dobro(a));
    printf ("%d\n", a);
    return 0;
}
```

# Parâmetros por referência e ponteiros

- Como vimos, a forma que usamos até aqui para expressar os parâmetros só permite a cópia de valores.
- Mas, e se quisermos que no exemplo anterior a alteração de `a` fosse visível para todo o programa?
- Para isso devemos declarar o parâmetro da função como um ponteiro.

# Parâmetros por referência e ponteiros

- Quais as saídas do seguinte código?

```
#include <stdio.h>

int dobro (int *a) {
    *a = *a * 2;
    return *a;
}

int main () {
    int a = 5;
    printf ("%d\n", a);
    printf ("%d\n", dobro(&a));
    printf ("%d\n", a);
    return 0;
}
```



# Ponteiros...

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

- Um ponteiro é uma variável que armazena endereço de memória.
- Sintaxe:

**tipo \*identificador;**

- A sintaxe é a mesma tanto para variáveis quanto para parâmetros

## Operando ponteiros:

- &
  - Retorna o endereço de uma variável
- \*
  - Na declaração, identifica a variável como um ponteiro.
  - Nas expressões e comandos, retorna o valor apontado pelo endereço armazenado no ponteiro.

# Ponteiros

```
#include <stdio.h>

int main () {
    int *p1;
    int a = 255;
    p1 = &a;
    printf ("%d\n", *p1);
    return 0;
}
```

Memória	
90000000	04
90000001	00
90000002	00
90000003	90
90000004	00
90000005	00
90000006	00
90000007	FF
90000008	00
90000009	00
9000000A	00
9000000B	00
9000000C	00

} int \*p1;

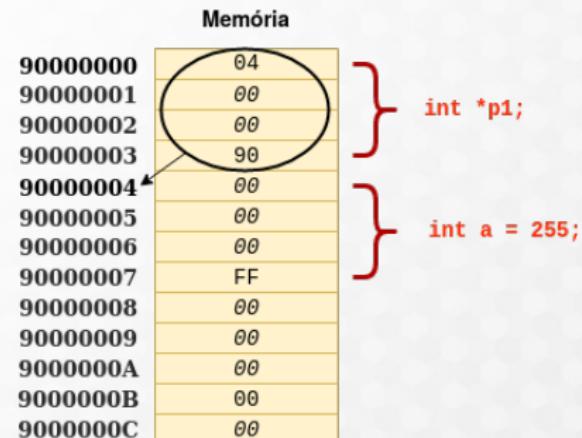
} int a = 255;



# Ponteiros

```
#include <stdio.h>

int main () {
    int *p1;
    int a = 255;
    p1 = &a;
    printf ("%d\n", *p1);
    return 0;
}
```



Quais os valores de a, b e c?

```
#include <stdio.h>

int main () {
    int *pt;
    int a = 5, b = 12, c = 19;

    pt = &a;
    b = *pt;
    *pt = c;
    printf ("%d %d %d", a, b, c);
    return 0;
}
```

Quais os valores de a, b e c?

```
#include <stdio.h>

int main () {
    int *pt;
    int a = 5, b = 12, c = 19;

    pt = &a; // pt aponta para o endereço de a
    b = *pt; // b recebe o valor presente no endereço
              // apontado por pt
    *pt = c; // o endereço apontado por pt recebe o
              // valor de c
    // Assim, a = c = 19 e b = 5
    return 0;
}
```

Qual o valor impresso?

```
#include <stdio.h>

int main () {
    int *pt1, *pt2;
    int a = 5, b = 5;

    pt1 = &a;
    pt2 = &b;

    printf("%c\n", pt1 == pt2 ? 'S' : 'N');
    return 0;
}
```

# Exercícios

Encontre os erros no código abaixo e aponte as devidas correções.

```
#include <stdio.h>
int main () {
    int x, *p;
    x = 100;
    p = x;
    printf("Valor de p: %d.\n", *p);
    return 0;
}
```

# Funções e ponteiros

- Voltando a pergunta inicial, quais as saídas do seguinte código?

```
#include <stdio.h>

int dobro (int *a) {
    *a = *a * 2;
    return *a;
}

int main () {
    int a = 5;
    printf ("%d\n", a);
    printf ("%d\n", dobro(&a));
    printf ("%d\n", a);
    return 0;
}
```

# Funções e ponteiros - Exemplo

Encontre os erros no código abaixo e aponte as devidas correções.

```
void troca (int *i, int *j) {  
    int *temp;  
    *temp = *i;  
    *i = *j;  
    *j = *temp;  
}
```

# Funções e ponteiros - Exemplo

Não inicializamos o endereço guardado por `*temp`, logo não sabemos para onde ele aponta

Para guardar o valor de `*i` precisamos de uma variável int, não um ponteiro para int

```
void troca (int *i, int *j) {  
    int temp;  
    temp = *i;  
    *i = *j;  
    *j = temp;  
}
```



- Ponteiros são extremamente poderosos e alteram diretamente a memória.
- Nunca desreferencie um ponteiro que você não sabe para onde aponta.
- Ponteiros não perguntam se o endereço guardado é válido.
- Voltaremos a discutir ponteiros quando estivermos tratando sobre vetores.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Problema

- Elabore um programa que recebe como entrada a idade de uma pessoa e diz se ela está apta ou não para votar.

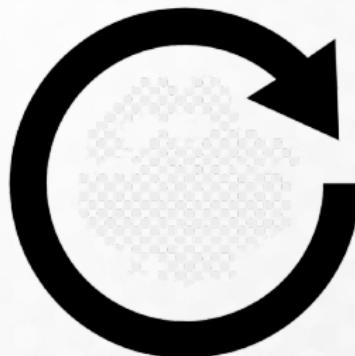
# Problema

- Elabore um programa que recebe como entrada a idade de uma pessoa e diz se ela está apta ou não para votar.
- E se o programa tiver que fornecer essa informação para 5 pessoas, o que poderíamos fazer?

# Problema

- Elabore um programa que recebe como entrada a idade de uma pessoa e diz se ela está apta ou não para votar.
- E se o programa tiver que fornecer essa informação para 5 pessoas, o que poderíamos fazer?
- E para 1000 pessoas?

# Estruturas de repetição

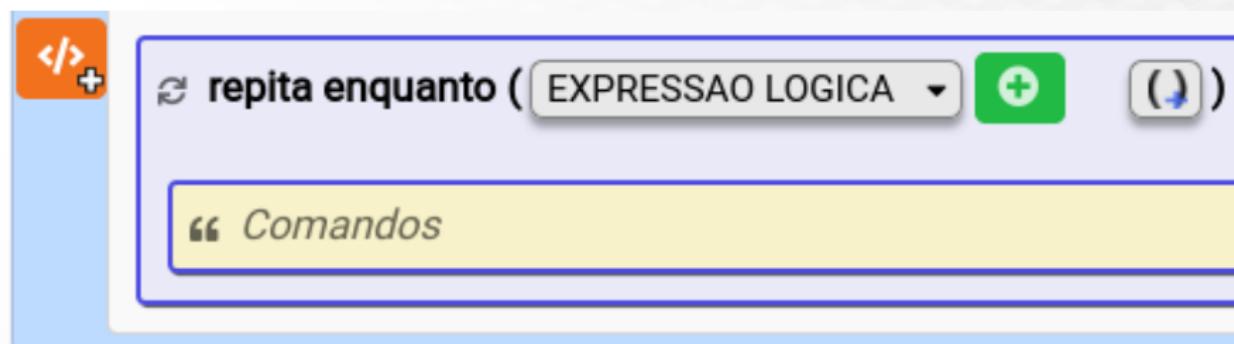


# Estruturas de repetição

- Estruturas de repetição ou **laços** são tipos de controle de fluxo.
- Permitem que trechos de código sejam executados **repetidamente** de acordo a **condição de controle** do laço.
- iVProg possui três estruturas de repetição **repita enquanto**, **repita até que falso** e **repita para**
- C possui três estruturas de repetição: **while**, **do...while** e **for**.

# Repita enquanto

- **repita enquanto** é um laço de repetição com **condição de entrada**.
  - Enquanto a condição for verdadeira executa os comandos internos
- **Sintaxe:**



# While

- **while** é um laço de repetição com **condição de entrada**.
- **Sintaxe:**

```
while (expressao_logica) {  
    comandos...  
}
```

## Exemplo de código

```
while (expressao_logica) {  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
}  
comando_fora_do_while;  
comando_fora_do_while;  
...
```



# While

```
while (expressao_logica) {  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
}  
comando_fora_do_while;  
comando_fora_do_while;  
...
```

A red curved arrow labeled "Falso" points from the closing brace of the while loop back to the condition. A green curved arrow labeled "Verdadeiro" points from the condition back to the opening brace of the loop. A green oval highlights the condition "expressao\_logica".

# While

Cuidado com laços infinitos!

```
#include <stdio.h>

int main () {
    while (1) {
        printf("Laco infinito!");
    }
    return 0;
}
```



# Exemplo

Crie um programa que imprima os primeiros 5 naturais.

# Exemplo

Crie um programa que imprima os primeiros 5 naturais.

- Resolver usando o iVProg.

# Exemplo

Crie um programa que imprima os primeiros 5 naturais.

- Resolver usando o iVProg.
- Resolver usando C.

# Exemplo

Crie um programa que receba um valor inteiro natural N. Em seguida, imprimir todos os valores de 0 até N.

# Exemplo

Crie um programa que receba um valor inteiro natural N. Em seguida, imprimir todos os valores de 0 até N.

- Resolver usando o iVProg.

# Exemplo

Crie um programa que receba um valor inteiro natural N. Em seguida, imprimir todos os valores de 0 até N.

- Resolver usando o iVProg.
- Resolver usando C.

# Do...While

- **do...while** é um laço de repetição com **condição de saída**.
- A condição de saída garante que o laço seja executado **pelo menos uma vez**.
- **Sintaxe:**

```
do {  
    comandos...  
} while (expressao_logica);
```

# Do...while

## Exemplo de código

```
do {  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
} while (expressao_logica);  
comando_fora_do_while;  
comando_fora_do_while;  
...
```

# Do...while

```
do {  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
    comando_dentro_do_while;  
} while (expressao_logica);  
comando_fora_do_while;  
comando_fora_do_while;  
...
```

A diagram illustrating the flow of control for a do...while loop. A green arrow points from the opening brace of the loop body to the start of the body. A green oval encloses the entire body of the loop, labeled "Verdadeiro" (True) at its right end. A red arrow points from the closing brace of the loop body back to the "while" keyword, which is also enclosed in a green oval labeled "expressao\_logica" (logical expression). A red arrow also points from the "while" keyword back to the "do" keyword.

Falso



IME-USP

# Exemplo

Crie um programa que leia um valor inteiro N e o imprima. Entretanto, faça com que o usuário seja obrigado a entrar com um número positivo.

- Quando o laço é interrompido?

# Exemplo

Crie um programa que leia um valor inteiro N e o imprima. Entretanto, faça com que o usuário seja obrigado a entrar com um número positivo.

- Quando o laço é interrompido?
- Resolver usando o iVProg.

# Exemplo

Crie um programa que leia um valor inteiro N e o imprima. Entretanto, faça com que o usuário seja obrigado a entrar com um número positivo.

- Quando o laço é interrompido?
- Resolver usando o iVProg.
- Resolver usando C.

# Exemplo

Resolva os exercícios no SAW

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Mais operadores!

- Operadores especiais de atribuição:

```
int main () {
    int x = 5;
    x += 1; // x = x + 1; // x++;
    x -= 1; // x = x - 1; // x--;
    x *= 2; // x = x * 2;
    x /= 2; // x = x / 2;
    x %= 5; // x = x % 5;
}
```



# Ordem de precedência

Tabela de precedência atualizada

Operadores
( ) ++ -- (sufixos)
! & ++ -- (prefixos)
* / %
+ -
< <= > >=
== !=
&&
?:
=
-= +=
*= /= %=



# Repita enquanto - iVProg

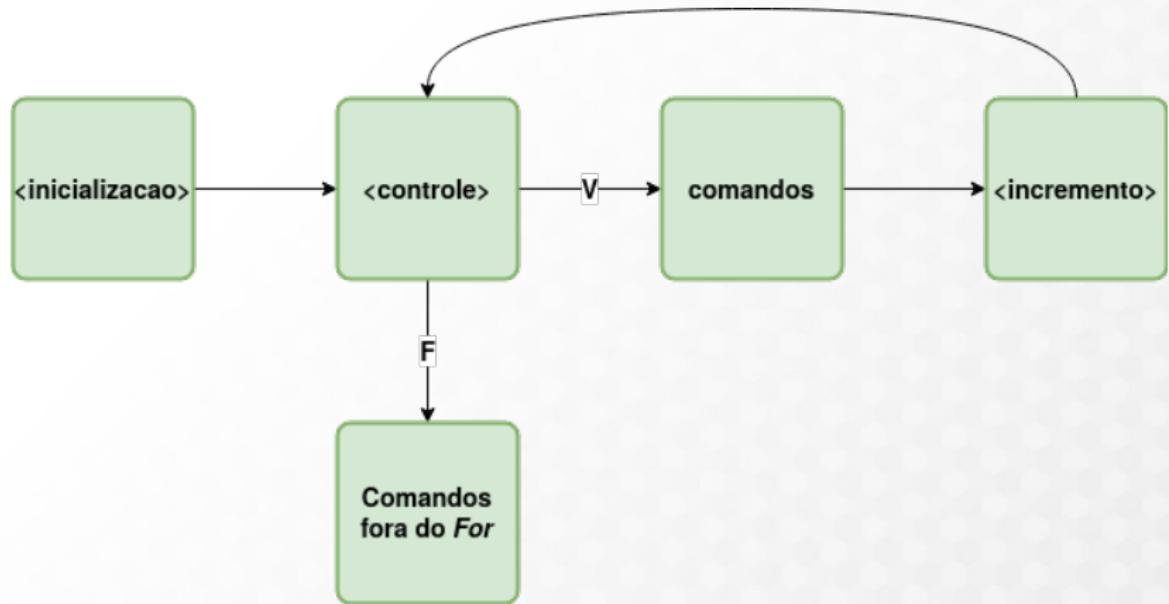
The screenshot shows the iVProg interface with a blue header bar containing icons for file operations, code, download, undo, redo, run, and help. Below the header is a toolbar with a purple 'x<sup>2</sup>' icon and a '+' icon. The main workspace displays a script starting with 'funcao vazio inicio ()'. Inside the script, there are two blocks: 'inteiro i <- 0' and 'inteiro limite <- 0'. Below these is a 'repita\_para' loop block with three parameters: 'inicialização' (containing 'i de 0'), 'controle' (containing 'até limite'), and 'incremento' (containing 'passo << + 1'). A lock icon is also present next to the loop block. At the bottom of the workspace is a green '+' button labeled 'Função'.

- **for** é um laço de repetição similar aos anteriores.
- Normalmente usado em contextos **iterativos**(navegar por uma série de valores).
- **Sintaxe:**

```
for (<inicializacao>; <controle>; <incremento>) {  
    comandos...  
}
```

- <**inicializacao**>: aqui se inicializa a variável de controle (iterador).
- <**controle**>: aqui inserimos uma expressão lógica que controla o número de repetições (quando termina?).
  - O **for** é executado enquanto a expressão presente em **controle** for verdadeira.
- <**incremento**>: aqui inserimos o comando que altera o valor da variável de controle a cada passo.
  - Um passo representa a execução de todos os comandos dentro do bloco **for**.

- A <inicializacao> é executada apenas uma vez, antes de executar os comandos do bloco.
- A expressão de <controle> é verificada todas as vezes antes de iniciar a execução do bloco
- O comando de <incremento> é executado todos as vezes assim que a execução do bloco é finalizada, antes da verificação da expressão de controle



- O C99 permite declarar variável dentro da <inicializacao>.
- Lembra do comando **break**? Ele serve para forçar a saída de qualquer laço de repetição.
- Temos também o comando **continue**, ele serve para pular um passo.

# Exemplo

O que o código abaixo faz?

```
#include <stdio.h>
int main(){
    int i;
    for (i=0; i<10; i++){
        printf("%d\n", i);
    }
}
```



IME-USP

# Exemplo

O que o código abaixo faz?

```
#include <stdio.h>
int main(){
    int i;
    for (i=0; i<10; i++){
        printf("%d\n", i);
    }
}
```

- Imprime inteiros de 0 a 9
- Faça o código em iVProg

# Problema

- Crie um programa para ler **n** valores inteiros e encontrar o maior deles.
  - Solicite ao usuário quantos valores serão digitados
  - Resolver em iVProg e em C

# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
0

Maior igual a 0 só funciona para os naturais.



# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
3

$3 > \text{Maior}$ , então troca.

# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
3

Não troca, pois  $2 < \text{Maior}$



# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
15

$15 > \text{Maior}$ , então troca



IME-USP

# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
15

Não troca, pois  $0 < \text{Maior}$



# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
94

$94 > \text{Maior}$ , então troca



IME-USP

# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
218

$218 > \text{Maior}$ , então troca



IME-USP

# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
218

Não troca, pois  $2 < \text{Maior}$



# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
218

Não troca, pois  $47 < \text{Maior}$



# Problema

Encontrar o maior (**simulação**)

Entradas	
1	3
2	2
3	15
4	0
5	94
6	218
7	2
8	47

Maior
218

Ao final, o maior valor lido estará em  
Maior



# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Problema

- Elaborar um programa que leia os preços de 5 itens e produza como saída o total da compra e a média dos preços.
- Entretanto, é preciso ler todos os preços no primeiro momento e só depois realizar os cálculos.
- Como resolver este problema?

# Problema

- Elaborar um programa que leia os preços de 5 itens e produza como saída o total da compra e a média dos preços.
- Entretanto, é preciso ler todos os preços no primeiro momento e só depois realizar os cálculos.
- Como resolver este problema?
- Criar 5 variáveis, uma para cada preço.

# Problema

- Elaborar um programa que leia uma quantidade N (máx. 30) de itens. Em seguida, deve ler o preço de cada um dos N itens. O programa deve ter como saída o total da compra e a média dos preços.
- Entretanto, é preciso ler todos os valores primeiro e só depois realizar os cálculos.
- Como resolver este problema?

# Problema

- Elaborar um programa que leia uma quantidade N (máx. 30) de itens. Em seguida, deve ler o preço de cada um dos N itens. O programa deve ter como saída o total da compra e a média dos preços.
- Entretanto, é preciso ler todos os valores primeiro e só depois realizar os cálculos.
- Como resolver este problema?
- Criar 30 variáveis para quantidades, 30 para preços e escrever o código de forma que utilize o número correto de variáveis?

# Array (Arranjo)

- Um *array* é uma coleção de itens armazenados de **forma sequencial** na memória.
- Tem como objetivo armazenar múltiplos valores de um mesmo tipo.
- Existem **arrays** de uma ou mais dimensões.
- Abordaremos primeiro os arrays de uma dimensão, também chamados **vetores**.
- **Arrays** possuem um número fixo de elementos.

## Declaração de um vetor

- **Sintaxe:**

```
tipo identificador[tamanho];
```

- **Exemplo:**

```
int valores[5];
```

```
int n = 20;  
int vetor[n];
```

- E se quisermos inicializar?

- Sintaxe:

```
tipo identificador[tamanho] = {/* lista de tamanho  
elementos*/};
```

```
tipo identificador[] = {/* lista de n elementos*/};
```

- Exemplo

```
int valores[5] = {3,2,1,0,4};  
int valores[] = {5,7,6,8,9};
```

- Como acessar os valores de um vetor? **Utilizando índices!**
  - Cada elemento do vetor possui um **índice**.
  - Índices são números inteiros, sendo 0 o índice do primeiro elemento e  $n-1$  é o índice do último elemento do vetor de tamanho  $n$ .
    - Ex: Para acessar os elementos de um vetor de tamanho 5, utilizamos os índices 0, 1, 2, 3 e 4.

```
#include <stdio.h>
int main () {
    int valores [5] = {3,2,1,0,4};
    printf ("%d ", valores [1]); // Imprime 2
    printf ("%d\n", valores [0]); // Imprime 3
    return 0;
}
```

# Vetor

- Declarando o vetor, de identificador "valores", com 5 posições e com a inicialização de seus valores:

```
int valores[5] = {13, 22, 18, 60, 4};
```

- Cada elemento do vetor possui um número de índice que permite seu acesso, tanto para atribuir um valor quanto para lê-lo:

	0	1	2	3	4
valores =	13	22	18	60	4



# Array em C

- É um tipo especial de ponteiro
  - Um ponteiro é um tipo de variável que armazena endereço de uma outra variável.
  - Um *array* armazena um conjunto de endereços igual ao seu tamanho.
- Não podem ser atribuídas umas às outras:
  - Representam endereços de memória. Uma atribuição não faria uma cópia, apenas criaria um novo apontador para aqueles endereços.

# Entrada/Saída com Vetores

- Como ler um valor para um vetor?

```
int vetor [5];
scanf("%d", &vetor [0]); // O operador & ainda é necessário
// A leitura é feita posição por posição.
```

- Como escrever o valor armazenado em um vetor?

```
int vetor [5] = {1,2,3,4,5};
printf("%d\n", vetor [4]);
// A escrita também é feita posição por posição.
```

- Como atribuir um valor para um determinado elemento do vetor?

```
int vetor [5];
vetor [0] = 19;
vetor [1] = 60;
```

# Tipos de arrays

- Arrays podem ser de qualquer tipo simples (int, float, double e char)
- Todos os elementos do array devem ser do mesmo tipo do array

```
int vetor1[5] = {0,1,2,3,4};  
float vetor2[3] = {1.5, 2.7, 7.3};  
double vetor3[4] = {1.6, 0.3, 5.6, 25.1};  
char vetor4[5] = {'a', 'b', 'c', 'd', 'e'};
```

- Vetores de char podem, com uma pequena inserção, ser tratados como Strings (texto)
  - Veremos mais a frente quando falarmos de Strings.

## Percorrendo todos os elementos do vetor

- Para percorrer o vetor completamente, é necessário utilizar um **comando de repetição** (*for* é o mais utilizado).
- Em geral, na forma

```
for(i=0; i<tamanho; i++) {  
    // código  
}
```

- Acessando o vetor na posição **i**

```
vetor[i]
```

## Percorrendo todos os elementos do vetor

```
int valores[5] = {13, 22, 18, 60, 4};
```



- Acessando e imprimindo todos os elementos do vetor:

```
for (int i = 0; i < 5; i++) {
    printf("%d", valores[i]);
}
```

- Recebendo valores da entrada e alocando-os em cada posição do vetor:

```
for (int i = 0; i < 5; i++) {
    scanf("%d", &valores[i]);
}
```

## Erros comuns ao manipular vetores

- Tentar acessar um índice além da capacidade do vetor.
  - Exemplo: em um vetor com 6 posições, o programador tentar acessar o elemento de índice 6 → vetor[6].
  - **No C** acessar uma posição fora do vetor é um erro lógico
  - O código compila e executa, porém, o resultado pode não ser o esperado
  - Não temos controle do que está em posições fora do vetor, então nunca sabemos o que terá naquelas posições

# Erros comuns ao manipular vetores

- Tentar inicializar todos os elementos de um vetor de uma só vez.

```
int vetor[10];
vetor = 0; // Errado
```

- O correto é utilizar uma estrutura de repetição `for` para atribuir o valor a cada elemento:

```
int vetor[10];
for(int i = 0; i < 10; i++) {
    vetor[i] = 0;
}
```

# Exercícios

Elaborar um programa que leia o preço de 5 itens. Em seguida receba também a quantidade comprada de cada item. O programa deve ter como saída o total da compra e a média dos preços unitários.

Entrada	Saída
20.60 59.00 2.50 17.70 30.50	646.20
4 5 6 4 6	26.06



# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Problema

- Precisamos implementar um programa para ler as notas de 4 provas de 50 alunos e calcular a média de cada aluno e a média da turma.
- Solução que temos em mãos: criar 4 vetores com 50 elementos cada, para cada uma das provas:

```
float prova1 [50], prova2 [50], prova3 [50],  
    prova4 [50], medias [50];
```

# Problema

- Precisamos implementar um programa para ler as notas de 4 provas de 50 alunos e calcular a média de cada aluno e a média da turma.
- Solução que temos em mãos: criar 4 vetores com 50 elementos cada, para cada uma das provas:

```
float prova1 [50], prova2 [50], prova3 [50],  
    prova4 [50], medias [50];
```

# Problema

Aluno	prova1	prova2	prova3	prova4	medias
0 Caroline	0 5.4	0 7.3	0 4.5	0 8.5	0 6.4
1 Debora	1 9.2	1 5.0	1 3.8	1 7.2	1 6.3
2 Matheus	2 3.8	2 8.2	2 9.4	2 6.7	2 7.0
... ...	... ...	... ...	... ...	... ...	... ...
48 Valdo	48 4.5	48 9.0	48 8.4	48 8.0	48 7.4
49 Wanessa	49 6.7	49 10.0	49 7.5	49 7.7	49 7.9

# Problema

- E se precisarmos armazenar 100 provas diferentes?

Aluno	prova1	prova2	prova3	prova4	medias
0 Caroline	0 5.4	0 7.3	0 4.5	0 8.5	0 6.4
1 Debora	1 9.2	1 5.0	1 3.8	1 7.2	1 6.3
2 Matheus	2 3.8	2 8.2	2 9.4	2 6.7	2 7.0
... ...	... ...	... ...	... ...	... ...	... ...
48 Valdo	48 4.5	48 9.0	48 8.4	48 8.0	48 7.4
49 Wanessa	49 6.7	49 10.0	49 7.5	49 7.7	49 7.9

# Problema

- Uma boa solução para resolver o problema é utilizar matrizes:

Aluno	0	1	2	...	98	99	medias
0 Caroline	0	5.4	7.3	4.5	...	8.5	5.6
1 Debora	1	9.2	5.0	3.8	...	7.2	9.8
2 Matheus	2	3.8	8.2	9.4	...	6.7	4.0
...	...	...	...	...	...	...	...
48 Valdo	48	4.5	9.0	8.4	...	8.0	6.6
49 Wanessa	49	6.7	10.0	7.5	...	7.7	9.0

# Matriz

- É um array de duas ou mais dimensões (multidimensionais).
- Podemos pensá-la como um vetor de vetores.
- O sistema reserva área de memória contígua, assim como os vetores.

# Matriz

- Sintaxe:

**tipo identificador [linhas] [colunas] ;**

**tipo identificador [n] [m] [k] ;**

# Matriz bi-dimensional

- Uma matriz bi-dimensional pode ser enxergada como uma tabela de m linhas e n colunas.

		colunas			
		0	1	2	3
linhas	0				
	1				
	2				

- As linhas e colunas são numeradas de 0 até o *tamanho - 1*.

# Matriz

```
int main () {
    unsigned char imgRGB [1920] [1080] [3];
    unsigned int grafo [100] [100];
    double vlrVendLojaMes [2000] [12];
    int matriz [20] [20] [20];
    return 0;
}
```

# Matriz

## Inicialização de matriz

```
int main () {
    int matriz [3] [4] = {{10 ,20 ,30 ,40} ,
                          {50 ,60 ,70 ,80} ,
                          {90 ,100 ,110 ,120}};
    return 0;
}
```

# Matriz

Inicialização de matriz: e se fossem grandes dimensões?

```
int main () {  
    int matriz[300][200];  
    return 0;  
}
```

# Matriz

Inicialização de matriz: e se fossem grandes dimensões?

```
int main () {  
    int matriz [300] [200];  
    return 0;  
}
```

Laço de repetição!

# Matriz

Acessando os elementos de uma matriz:

<variável>[<indice\_linha>][<indice\_coluna>]

```
// imprime o elemento da linha 2 e coluna 16 da matriz  
// notas:  
printf("%.2f", notas[2][16]);  
  
// multiplica a posição (i, j) da matriz 'mat' por 5:  
mat[i][j] = mat[i][j] * 5;
```



# Matriz

Leitura dos dados da matriz

Considere uma matriz ( $M \times N$ )

```
for (int i = 0; i < M; i++) { //laço para as linhas
    for (int j = 0; j < N; j++) { //laço para as colunas
        scanf("%d", &matriz[i][j]);
    }
}
```

# Matriz

## Impressão do conteúdo da matriz

Considere uma matriz ( $M \times N$ )

```
for (int i = 0; i < M; i++) { //laço para as linhas
    for (int j = 0; j < N; j++) { //laço para as colunas
        printf("%d", matriz[i][j]);
    }
    printf("\n"); //quebra a linha na saída
}
```

# Matriz

Inicializando cada elemento da matriz

Considere uma matriz ( $M \times N$ )

```
for (int i = 0; i < M; i++) //laço para as linhas
    for (int j = 0; j < N; j++)
        matriz[i][j] = 0;
```

# Exercício para resolver

Elaborar um programa para:

- Ler a **média bimestral** de 3 alunos.
- Calcular a **média anual** de cada aluno.
- Exibir em forma tabulada.
- Obs. : As médias serão informadas na mesma ordem.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Dado textual em C

- Em C, utilizamos o tipo **char** para armazenar uma representação dos caracteres:
  - Ex: A, B, a, g, 1, 2, ?, ...
- Efetivamente, um **char** é uma valor inteiro de 16 bits que assume os valores presentes na tabela ASCII.
- Utilizamos aspas simples ' ' para representar um valor do tipo char.

# Dado textual em C

- A entrada/saída do tipo **char**

```
#include <stdio.h>

int main () {
    char letra = 'g';
    // %c é o formatador tanto pra leitura quanto pra
    // escrita do tipo char
    scanf("%c", &letra);
    printf("%c", letra);
    return 0;
}
```

# Dado textual em C

	00	16	32	48	64	80	96	112
0	<b>NUL DLE</b>			0	@	P	'	p
1	<b>SOH DC1</b>	!	1	A	Q	a	q	
2	<b>STX DC2</b>	"	2	B	R	b	r	
3	<b>ETX DC3</b>	#	3	C	S	c	s	
4	<b>EOT DC4</b>	\$	4	D	T	d	t	
5	<b>ENQ NAK</b>	%	5	E	U	e	u	
6	<b>ACK SYN</b>	&	6	F	V	f	v	
7	<b>BEL ETB</b>	'	7	G	W	g	w	
8	<b>BS CAN</b>	(	8	H	X	h	x	
9	<b>HT EM</b>	)	9	I	Y	i	y	
10	<b>LF SUB</b>	*	:	J	Z	j	z	
11	<b>VT ESC</b>	+	;	K	[	k	{	
12	<b>FF FS</b>	,	<	L	\	l		
13	<b>CR GS</b>	-	=	M	I	m	}	
14	<b>SO RS</b>	.	>	N	^	n	~	
15	<b>SI US</b>	/	?	O	-	o	DEL	

# Dado textual em C

- Assim como os tipos numéricos: int, float, double, long, short, temos vetores do tipo **char**:
  - Entretanto, existe em C uma forma particular dos vetores de char que chamamos de String

```
#include <stdio.h>

int main () {
    //String sempre entre aspas " "
    char texto[50] = "Eu sou uma string em C!";
    //Vetor padrão
    char letras[12] = {'S', 'o', 'u', ' ', 's', 't', 'r',
                      'i', 'n', 'g', '?'};
    return 0;
}
```

# Dado textual em C

- Assim como os tipos numéricos: int, float, double, long, short, temos vetores do tipo **char**:
  - Entretanto, existe em C uma forma particular dos vetores de char que chamamos de String

```
#include <stdio.h>

int main () {
    //String sempre entre aspas " "
    char texto[50] = "Eu sou uma string em C!";
    //Vetor padrão
    char letras[12] = {'S', 'o', 'u', ' ', 's', 't', 'r',
                      'i', 'n', 'g', '?'};
    return 0;
}
```

# Dado textual em C

- Strings são um tipo especial de vetor.
- Strings possuem um formatador especial: **%s**
- Exemplo:

```
#include <stdio.h>

int main () {
    //String sempre entre aspas " "
    char texto[50] = "Eu sou uma string em C!";
    //
    printf("%s\n", texto);
    return 0;
}
```



# Dado textual em C

- Se todo dado não inicializado fica com o valor do lixo da memória, como a frase foi impressa corretamente?
- Sorte?

```
#include <stdio.h>

int main () {
    //String sempre entre aspas " "
    char texto[50] = "Eu sou uma string em C!";
    //
    printf("%s\n", texto);
    return 0;
}
```



## Dado textual em C

- Toda String em C termina com o caractere nulo - '\0' - da tabela ASCII
- Ao usar aspas duplas, o compilador insere esse caractere automaticamente!
- Podemos também inserir manualmente

```
#include <stdio.h>

int main () {
    // Caractere nulo inserido manualmente
    char string[] = {'S', 'o', 'u', ' ', 's', 't', 'r',
                     'i', 'n', 'g', '\0'};
    printf("%s\n", string);
    return 0;
```

# Dado textual em C

0	1	2	3	4	5	6	7	8	9
'o'	'l'	'a'		'M'	'u'	'n'	'd'	'o'	'\0'

O caractere nulo indica onde a string termina e é crucial para as funções de manipulação de strings. Além disso, ele conta para o total do tamanho da string.

# Dado textual em C

```
#include <stdio.h>

int main () {
    char letras[12] = {'S', 'o', 'u', ' ', 's', 't', 'r',
                      'i', 'n', 'g', '?'};
    printf("%s\n", letras);
    return 0;
}
```

## ● Atenção

- O código acima funciona uma vez que o compilador completa o vetor com o caractere nulo.
- Preferencialmente, SEMPRE coloque o '\0' para evitar problemas!

# Dado textual em C

0	1	2	3	4	5	6	7	8	9	10
's'	'o'	'u'		's'	't'	'r'	'i'	'n'	'g'	'?'

Vetor de char completamente preenchido sem ser terminado por '\0' não é uma string!



## Entrada/Saída

- Se quisermos ler uma string utilizamos o formatador "%s":

```
scanf("%s", &string);
```

- &string ao invés de &string[indice] por que a leitura nesse caso é feita na totalidade.
- Se quisermos ler um caractere apenas utilizamos o formatador "%c":

```
scanf("%c", &string[0]);
```

- Neste caso é obrigatório fornecer a posição de leitura.
- A escrita é de forma análoga para ambos os casos:

```
printf("%s", string);
```

- **Atenção: é obrigatório que string termine com o caractere nulo.**

```
printf("%c", string[0]);
```

- Imprime o caractere presente no índice informado

## Riscos de segurança

- Utilizar **scanf** para leitura de strings não é seguro.
- Permite uma entrada maior que o declarado, corrompendo a memória.
- Pode ser uma porta de ataque ao sistema.
- Utilize a função **fgets**:

```
fgets(<variavel>, <tamanho>, <origem>);
```

```
#include <stdio.h>

int main () {
    char letras[21];
    fgets(letras, 21, stdin);
    // letras - variavel que ira armazenar a string lida
    // 21 - total de caracteres que se pode armazenar,
    // incluindo o '\0'
    // stdin - fonte dos dados, entrada padrao
    return 0;
}
```

# Exercício para resolver juntos

Elaborar um programa que receba uma string e um caractere. Em seguida, encontre o total de ocorrências do caractere no conteúdo da string.

Entrada	Saída
"ordem" 'o'	1
"caractere" 'c'	2
"verao" 'w'	0



IME-USP

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Operações com strings

- Em C, temos um conjunto de funções pré-definidas para manipulação de cadeias de caracteres (**string**)
  - É necessário incluir a biblioteca `<string.h>` no seu código.

```
#include <stdio.h>
#include <string.h>

int main () {
    return 0;
}
```

# Operações com strings

- Cópia de strings - **strcpy**

```
strcpy(s, r)
```

- A função copia o conteúdo da string r para a string s.
- Atenção: a string s deve ter no mínimo o tamanho de  $r + 1$

```
#include <stdio.h>
#include <string.h>

int main () {
    char cadeia_1[] = "O pe de feijao";
    char cadeia_2[30];
    strcpy(cadeia_2, cadeia_1);
    printf("%s\n", cadeia_2);
    printf("%s\n", cadeia_1);
    return 0;
}
```

# Operações com strings

- Concatenação de strings - **strcat**

`strcat(s, r)`

- A função concatena o conteúdo da string r no final da string s.
- Atenção: a string s deve possuir espaço suficiente para armazenar seu próprio conteúdo mais o conteúdo de r + 1

```
#include <stdio.h>
#include <string.h>

int main () {
    char cadeia_1 [] = "o pe de feijao";
    char cadeia_2 [30] = "Joao e ";
    strcat(cadeia_2, cadeia_1);
    printf ("%s\n", cadeia_2);
    printf ("%s\n", cadeia_1);
    return 0;
}
```

# Operações com strings

- Comparação de strings - **strcmp**

strcmp(s, r)

- A função compara a string s com a string r.

- **Retorna:** inteiro < 0, caso r > s; inteiro > 0, caso s > r; inteiro == 0, caso s == r.

```
#include <stdio.h>
#include <string.h>

int main () {
    char cadeia_1 [] = "abcdef";
    char cadeia_2 [30] = "ABCDEF";
    int retorno;
    retorno = strcmp(cadeia_1, cadeia_2);
    if (retorno == 0)
        printf("São iguais!\n");
    else
        printf("São diferentes!\n");
    return 0;
}
```

# Operações com strings

- Tamanho de uma string - **strlen**

`strlen(s)`

- A função calcula e retorna o tamanho de s.

```
#include <stdio.h>
#include <string.h>

int main () {
    char cadeia_1 [] = "Olá mundo!";
    int tamanho;
    tamanho = strlen(cadeia_1);
    printf("%d\n", tamanho);
    return 0;
}
```



IME-USP

## Exercício para resolver juntos

Elaborar um programa que receba duas strings, S1 e S2, com tamanho máximo de 20 caracteres cada. Implemente os seguintes itens:

- Imprimir o tamanho da string S1.
- Comparar a string S1 com a string S2 e imprimir o resultado dessa comparação.
- Concatenar a string S1 com a S2 e imprimir o resultado da concatenação.
- Imprimir a string S1 de forma reversa.
- Imprimir S1 e S2 em ordem alfabética, considere que ambas estão em maiúsculo.

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Fluxos de dados

- Até agora utilizamos fgets usando a entrada *stdin*
- *stdin* é um exemplo de fluxo(stream) de dados padrão
- Sendo os mais comuns:
  - **stdin**: Fluxo padrão de entrada, em geral, entrada do teclado
  - **stdout**: Fluxo padrão de saída, em geral, onde o programa escreve
  - **stderr**: Fluxo padrão de erros: usado para mensagens de erro e diagnóstico do sistema



# Fluxos de dados

- **stdin** é apenas um arquivo
- Os fluxos de dados são arquivos que o C deixa aberto e pronto para serem lidos
- Então a função fgets pode ser definida como:
  - **fgets(char \*str, int tamanho, FILE \*fp)**
  - **fp** é um apontador para um arquivo
- Podemos utilizar outros arquivos? Quais as vantagens?



# Arquivos

- Arquivos permitem manter os dados na memória secundária, de maneira permanente.
- Os dados em arquivos podem ser acessados de forma não sequencial.
- Mais de um programa pode acessar os dados ao mesmo tempo.

# Tipos de arquivos

- Podemos manipular dois tipos principais de arquivos: de texto e binários.
- Arquivo texto:
  - armazena caracteres que podem ser exibidos diretamente na tela ou modificados por um editor de textos, como o bloco de notas.
  - cada caractere armazenado ocupa 8 bits.

# Tipos de arquivos

- Podemos manipular dois tipos principais de arquivos: de texto e binários.
- Arquivo binário:
  - armazena uma sequência de bits, de acordo com os padrões dos programas que o gerou. Por exemplo: arquivos executáveis, arquivos compactados, etc.

# Manipulação de arquivos em C

- Existe um conjunto de funções para a manipulação de arquivos em C, reunidos na biblioteca padrão de entrada e saída: *stdio.h*.

# Manipulação de arquivos em C

- A linguagem C não possui função que leia todas as informações de um arquivo automaticamente.
- As funções disponíveis se limitam a abrir/fechar e ler/escrever caracteres/bytes.
- A leitura/escrita será responsabilidade do programador para cada situação específica.

# Manipulação de arquivos em C

- Todas as funções de manipulação de arquivos trabalham com o conceito de "ponteiro de arquivo". Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
1      FILE *arq;
```

- **arq** é o ponteiro para arquivos que nos permitirá manipular arquivos no C.

# Manipulação de arquivos em C

- Passos:

- 1 Criar um ponteiro para a estrutura FILE.
- 2 Abrir o arquivo.
- 3 Ler ou gravar dados no arquivo.
- 4 Fechar o arquivo.

# Abrindo um arquivo

- Para usar um arquivo em C é necessário abri-lo.
- Para a abertura de um arquivo, usa-se a função **fopen**:

```
1     arq = fopen("ArqGrav.txt", "rt");
```

- Parâmetros:

- nome do arquivo
- modo de abertura

# Abrindo um arquivo

- No parâmetro para o nome do arquivo, pode-se trabalhar com caminhos absolutos ou relativos.
- **Caminho absoluto:** descrição de um caminho desde o diretório raiz.
  - C:\\Projetos\\\\dados.txt
  - /home/user/Documents/dados.txt
- **Caminho relativo:** descrição de um caminho desde o diretório corrente (onde o programa está salvo)
  - dados.txt
  - ./dados.txt
  - diretorio/dados.txt

# Abrindo um arquivo

- O modo de abertura determina que tipo de uso será feito do arquivo.
- A tabela a seguir mostra os modos válidos de abertura de um arquivo.

# Abrindo um arquivo

Modo	Arquivo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"rb"	Binário	Leitura. Arquivo deve existir.
"wb"	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"ab"	Binário	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+"	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+"	Texto	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir
"a+"	Texto	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+b"	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+b"	Binário	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+b"	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").

# Abrindo um arquivo

- Abrindo um arquivo "arq1.txt" no diretório "dados", para leitura e escrita:

```
int main() {
    FILE *arquivo;
    arquivo = fopen("dados/arq1.txt", "a+");

    if (arquivo == NULL) {
        printf("Não foi possível abrir o arquivo!");
        system("pause");
        exit(1);
    }

    fclose(arquivo);

    return 0;
}
```

## Fechando um arquivo

- Sempre que terminamos de usar um arquivo que abrimos, devemos fechá-lo. Para isso usa-se a função **fclose()**.
- O ponteiro **arquivo** passado à função **fclose()** determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

# Escrita/Leitura em Arquivos

- A linguagem C conta com uma série de funções de leitura/escrita que variam de funcionalidade para atender as diversas aplicações.
- Existem funções na linguagem C que permitem ler/escrever uma sequência de caracteres, isto é, uma string:
  - **fputs()**
  - **fgets()**

# Escrita em arquivos

- **fputs()**: escreve uma string no arquivo.

```
int main() {
    FILE *arquivo;
    char texto[30] = "Hello world!";
    arquivo = fopen("dados/arq1.txt", "a+");
    if (arquivo == NULL) {
        printf("Nao foi possivel abrir o arquivo!\n");
        system("pause");
        exit(1);
    }
    fputs(texto, arquivo);
    fclose(arquivo);
    return 0;
}
```

## Escrita em arquivos

- fputs() permite a escrita de strings, mas se precisarmos guardar outros tipos de dados, como números? fprintf()

```
int main() {
    FILE *arquivo;
    int idade = 59;
    arquivo = fopen("dados/arq1.txt", "a+");
    if (arquivo == NULL) {
        printf("Nao foi possivel abrir o arquivo!\n");
        system("pause");
        exit(1);
    }
    fprintf(arquivo, "%d", idade);
    fclose(arquivo);
    return 0;
}
```

# Leitura de arquivos

- Para ler uma string de um arquivo, podemos usar a função `fgets()`.
- `fgets()` lê a string até que um caractere de nova linha seja lido ou *tamanho-1* caracteres tenham sido lidos.
- Em caso de erro ou fim de arquivo, a função `fgets` retorna NULL.

# Leitura de arquivos

- Lendo a primeira linha do arquivo:

```
int main() {  
    FILE *arquivo;  
    arquivo = fopen("dados/arq1.txt", "a+");  
    char texto[100];  
  
    if (arquivo == NULL) {  
        printf("Nao foi possivel abrir o arquivo!\n");  
        system("pause");  
        exit(1);  
    }  
  
    fgets(texto, 100, arquivo);  
    printf("%s", texto);  
  
    fclose(arquivo);  
  
    system("pause");  
  
    return 0;  
}
```

# Leitura de arquivos

- Lendo todas as linhas do arquivo:

```
int main() {
    FILE *arquivo;
    arquivo = fopen("dados/arq1.txt", "a+");
    char texto[100];

    if (arquivo == NULL) {
        printf("Nao foi possivel abrir o arquivo!\n");
        system("pause");
        exit(1);
    }

    while(fgets(texto, 100, arquivo) != NULL) {
        printf("%s\n", texto);
    }

    fclose(arquivo);

    system("pause");

    return 0;
}
```

# Leitura de arquivos

- A função **fgets** permite a leitura de uma string e se precisarmos ler um valor inteiro de um arquivo? **fscanf()**

# Leitura de arquivos

```
int main() {
    FILE *arquivo;
    arquivo = fopen("dados/arq1.txt", "a+");
    int numero;

    if (arquivo == NULL) {
        printf("Nao foi possivel abrir o arquivo!\n");
        system("pause");
        exit(1);
    }

    fscanf(arquivo, "%d", &numero);

    printf("%d", numero);

    fclose(arquivo);

    system("pause");

    return 0;
}
```

# Leitura de arquivos

- Como ler um arquivo que não sabemos o tamanho?
- Como saber que chegamos ao final do arquivo?

# Leitura de arquivos

- Todo arquivo finaliza com um caractere específico - **EOF**
- Podemos verificar se a próxima leitura leria este arquivo
- **fgetc** (arquivo) == EOF
- **feof** (arquivo)

# Apagando um arquivo

- Existe uma função que permite apagar um arquivo do disco.
- Cuidado! O arquivo não irá para a "lixeira", será removido definitivamente.

```
1 remove("arquivo.txt");
```

- Informamos o caminho do arquivo, de forma relativa ou absoluta.
- A função remove devolve 0 se o arquivo for excluído com sucesso.

# Apagando um arquivo

```
int main() {
    int resultado;
    resultado = remove("dados/arq1.txt");

    if (resultado == 0) {
        printf("Arquivo removido com sucesso!\n");
    } else {
        printf("Não foi possível remover o arquivo, tente
               novamente!\n");
    }

    system("pause");

    return 0;
}
```

# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Alocação dinâmica de memória

- É possível alocar a quantidade desejada de memória, no momento desejado?
  - Não sei quantos alunos uma escola tem
    - O meu sistema roda em escolas diferentes

# Alocação dinâmica de memória

- Os espaços são alocados durante a execução do programa.
  - Conforme cada programa necessitar de memória, alocará os espaços.
  - Não há reserva antecipada de espaço.
- Solução: Dimensionamento de espaço.
- Utilizar ponteiro.
  - O sistema nos disponibiliza um bloco.
  - Apontamos para este bloco.

# Alocação dinâmica de memória - espaço

- Para alocarmos devemos dizer para o C quantos bytes queremos alocar
- Como saber quantos bytes queremos?
- Função **sizeof (tipo-ou-identificador)**:
  - Retorna o tamanho em bytes que uma variável do tipo ocupa
  - Exemplos:
    - `sizeof(char) = 1`
    - `sizeof(int) = 4`
    - `int vetor[4]; sizeof(vetor) = 16`



# Alocação dinâmica de memória - funções

- Funções úteis:
  - malloc
  - realloc
  - calloc
  - free
- Estas funções estão na **stdlib.h**

# Alocação dinâmica de memória - **malloc**

- Abreviatura de *memory allocation*.
- Aloca **n** bytes
- Sintaxe:
  - Retorna um ponteiro.
  - Retorna **NULL** se não alocou.
  - Recebe **size** em bytes

```
1 void *malloc(size_t size);
```

# Alocação dinâmica de memória - malloc

- Exemplo

```
int main() {
    int *vector = NULL; /* declaração do ponteiro */
    vector = (int*) malloc(25 * sizeof(int)); /* alocação
        de memória para o vector */
    vector[10] = 34; /* altera o valor da posição dez
        para trinta e quatro */
    free(vector); /* liberta a área de memória alocada
        */
    return 0;
}
```

# Alocação dinâmica de memória - `realloc`

- Às vezes é necessário alterar, durante a execução do programa, o tamanho de um bloco de bytes que foi alocado por `malloc`.
- A função `realloc` aumenta ou diminui o tamanho do bloco de memória especificado, movendo-o se necessário.
- Sintaxe:

```
1 void *realloc(void *p, size_t size);
```

- Recebe um apontador **p** para o bloco à ser modificado.
- Recebe **size** em bytes (novo tamanho).
- Retorna um ponteiro para bloco realocado.
- Retorna **NULL** se não alocou.



IME-USP

# Alocação dinâmica de memória - **realloc**

- Exemplo

```
int main() {
    int *vector = NULL; /* declaração do ponteiro */
    vector = (int*) malloc(1000 * sizeof(int)); /* aloca
        ção de memória para o vector */
    for(int i = 0; i < 990; i++) /* preenche o vetor*/
        scanf("%d", &vetor[i]);
    vector = realloc(v, 2000 * sizeof(int)); /* 
        redimensionamento de memória*/
    for(int i = 990; i < 2000; i++) /* preenche o vetor*/
        scanf("%d", &vetor[i]);
    free(vector); /* liberta a área de memória alocada
        */
    return 0;
}
```



# Alocação dinâmica de memória - **calloc**

- Aloca **n** espaços de tamanho **size** bytes. Inicializa todos os espaços com zero.
- Sintaxe:

```
1 void *calloc(tipo qtd_elementos, tipo tamanhoElemento);
```

- Recebe **qtd\_elementos**, quantidade de espaços à serem alocados.
- Recebe **tamanhoElemento**, tamanho de cada elemento em bytes.
- Retorna **NULL** se não alocou.
- Retorna um ponteiro para o bloco alocado.

# Alocação dinâmica de memória - **calloc**

- Exemplo

```
int main() {  
    char *caractere;  
    caractere = calloc (1, sizeof(char));  
    scanf ("%c", caractere);  
    return 0;  
}
```

# Desalocação de memória

- Como o computador sabe qual espaço de memória pode ser usado para uma nova variável?

# Desalocação de memória

- Como o computador sabe qual espaço de memória pode ser usado para uma nova variável?
- Uma tabela dizendo quais espaços cada variável ocupa!!!

# Desalocação de memória

- Como o computador sabe qual espaço de memória pode ser usado para uma nova variável?
- Uma tabela dizendo quais espaços cada variável ocupa!!!
- Ao alocar uma variável ela é inserida na tabela junto aos espaços utilizados
- Ao ser desalocada apagamos a entrada na tabela!

# Desalocação de memória - **free**

- Espaços não mais ocupados são liberados automaticamente em C?
  - Alocação **estática** - Sim
    - Ao encerrar o escopo, as variáveis daquele escopo são desalocadas
  - Alocação **dinâmica** - Não
    - O programador é o responsável por alocações dinâmicas
    - Solução: Utilizar a função **free**.

# Desalocação de memória - **free**

- Libera um espaço de memória que não está mais sendo utilizado
- Uma vez desalocada a memória pode ser associada à uma nova variável

1    **free**(**void** \***p**)

- Recebe **p**, ponteiro para a memória que será desalocada

# Desalocação de memória - Cuidados

- Devemos sempre desalocar um espaço de memória que não será mais utilizado
  - Não desalocar espaços “inúteis” causa **Memory Leak**
  - Programas que consomem cada vez mais memória e ficam cada vez mais lentos.
- Uma vez desalocado o espaço os valores ainda continuam lá
  - **free** não limpa os espaços
  - Não devemos utilizar espaços desalocados sob risco de causa instabilidade no programa



# Dúvidas?

# Estruturas/Registros

- É um "pacote" de variáveis, possivelmente de tipos diferentes.
- Os dados agrupados na struct são denominados **campos(fields)**.
- Structs são muito usadas quando temos elementos em nossos programas que precisam e fazem uso de vários tipos de variáveis e características.

# Estruturas/Registros



# Estruturas/Registros

## Sintaxe

```
int main () {
    struct identificador_struct {
        <tipo1> <identificador1>;
        <tipo2> <identificador2>;
        ...
        <tipoN> <identificadorN>;
    };
}

/*Sintaxe de declaração da variável do "tipo" definido*/
    struct identificador_struct variavel;
    return 0;
}
```

# Estruturas/Registros

```
int main() {
    struct endereco {
        char *cep;
        char *bairro;
        char *rua;
        int numero;
    };

    struct endereco x; // um registro x do tipo
    endereco

    x.cep = "05508090";
    x.bairro = "Butantã";
    x.rua = "Rua do Matão";
    x.numero = 1010;

    return 0;
}
```

# Estruturas/Registros - Typedef

- Você pode definir os seus próprios tipos-de-dados recorrendo ao **typedef**!
- A palavra reservada **typedef** nada mais é do que um atalho em C para que possamos nos referir a um determinado tipo existente com nomes sinônimos.

# Estruturas/Registros - Typedef

```
#include <stdio.h>
#include <string.h>

struct Livros {
    char titulo[50];
    char autor[50];
    char assunto[100];
    int livro_id;
};

typedef struct Livros exemplar;
```

# Estruturas/Registros - Typedef

```
#include <stdio.h>
#include <string.h>

typedef struct {
    char titulo[50];
    char autor[50];
    char assunto[100];
    int livro_id;
} Livros;
```

# Estruturas/Registros - Typedef

```
int main(){

    exemplar livro;

    strcpy(livro.titulo, "C completo e total");
    strcpy(livro.autor, "Herbert Schildt");
    strcpy(livro.assunto, "Tutorial de programação em C");
    livro.livro_id = 1234;

    printf ("Título do Livro: %s\n", livro.titulo);
    printf ("Autor do Livro: %s\n", livro.autor);
    printf ("Assunto do Livro: %s\n", livro.assunto);
    printf ("ID do Livro: %d\n", livro.livro_id);

    return 0;
}
```



# Estruturas/Registros - Observações

- Structs:
  - NÃO são classes
  - Apenas agrupam dados
  - Podem ter campos de quaisquer tipos, inclusive de outra Struct



# Dúvidas?

# Introdução à Programação

Bernardo Martins Ferreira

Instituto de Matemática e Estatística  
Universidade de São Paulo  
São Paulo - SP

Verão 2023

# Aprendemos a programar e agora?

- Front-end: Interface entre usuário e sistema
- Back-end: Interface entre sistemas
- Full Stack: Front-end e Back-end
- Cientista de dados: Trata dados brutos em uma forma interpretável
- Devops, Tech Recruiter, Desenvolvedor de Jogos, Analista de segurança, etc.

# O que aprender?

- Tópicos de programação
  - Estrutura de dados
  - Programação orientada a objetos
- Outras linguagens
  - Python
  - Javascript
  - Java

# O que aprender?

- Expressões regulares
- Métodos ágeis
- Rede de computadores

# Tópicos de programação - Estrutura de dados

- Formas de guardar e organizar dados
  - Pilha
  - Fila
  - Lista
  - Árvore
- Banco de dados
- Disciplina Tópicos de programação no verão

# Programação orientada a Objetos (POO)

- Utiliza classes de objetos para modelar um problema
- Classes são instanciadas em objetos
- Ex: classe animal pode ter a instância cachorro
- Facilita a implementação de problemas reais
- C não permite orientação a objetos

# Linguagens

- É importante conhecer várias linguagens
- Linguagens diferentes são melhores em contextos diferentes
  - Ex: Javascript é mais utilizada para web enquanto python é para ciência de dados
- Projetos utilizam diversas linguagens para funcionar
  - ex: Javascript na aplicação, PHP no servidor e sql no banco de dados.
- Ainda é uma área muito nova, qualquer linguagem pode cair em desuso

# Linguagens - Python

- Sintaxe simples
- Possui muitas bibliotecas e frameworks
- Muito utilizada em ciência de dados, aprendizado de máquina e inteligência artificial
- Algumas bibliotecas: Pandas, scikit learn, matplotlib, Tensorflow...

# Linguagens - Javascript

- Linguagem mais popular para desenvolvimento web
- Alta demanda
- Quase obrigatória para qualquer programador
- frameworks e bibliotecas: nodeJS, React, angular, vue, ...

- Uma das linguagens mais robustas
- Está sempre no topo das linguagens mais utilizadas
- Em geral usada em aplicações cliente-servidor
- Orientada a objetos
- Aplicações mobile, desktop, web, inteligência artificial, cloud...

# Linguagens - Kotlin/Swift/React Native

- Desenvolvimento de aplicativos
- Kotlin: multiplataforma (iOS e Android), compila para JVM
- Swift: proprietária da Apple (iOS)
- React Native: biblioteca javascript para desenvolvimento de aplicativos multiplataforma

# Linguagens - PHP

- Linguagem muito utilizada em servidores
- Rápida e segura
- Biblioteca extensa
- Composer, Laravel, codeigniter, Symphony
- O servidor do moodle é PHP

# Outras Linguagens

- Typescript
- C++
- C#
- GO

# Expressões regulares

- Padrões para reconhecimento de strings
- Ferramenta obrigatória para qualquer programador
- Facilita transformações e buscas em textos
- Aplicação: Ferramenta de busca em navegadores e sistemas operacionais

# Métodos ágeis

- Processo de engenharia de software para trabalho em grupos
- Facilita a interação entre membros da equipe
- Prevê mudanças nos requisitos
- Scrum, Kanban, Extreme Programming(XP)

# Redes de computadores

- Essencial para quem trabalha com a web
- Interação cliente-servidor e servidor-servidor
- Segurança
- Protocolo HTTP/HTTPS
- APIs Rest

- Ferramenta de compartilhamento de código
- Mantém histórico de mudanças no código
- Ótima maneira de expor projetos
- Servidores: **Github**, GitLab, Bitbucket, gogs, etc.

# Observações

- Aprenda linguagens e não frameworks/bibliotecas
- Exponha projetos no Github
- Se possível contribua com códigos open-source
- Computação é uma área **MUITO** extensa, é impossível ser bom em tudo
- De início foque em saber **bem** uma linguagem

# Como estudar?

- Faça projetos - comece simples
- Leia documentação das ferramentas que você está utilizando
- Não negligencie a teoria, um bom programador deve saber como as coisas funcionam
- Use plataformas de cursos
  - Udemy, Coursera, Alura, Khan Academy, 365datascience, Youtube, etc.



Muito obrigado por participar do  
curso!!