Processando a Informação: um livro prático de programação independente de linguagem

Rogério Perino de Oliveira Neves

Francisco de Assis Zampirolli

EDUFABC
editora.ufabc.edu.br

Notas de Aulas inspiradas no livro

Utilizando a(s) Linguagem(ns) de Programação:

C

Exemplos adaptados para Correção Automática no Moodle+VPL

Francisco de Assis Zampirolli

20 de novembro de 2022

Sumário

1 Processando a Informação: Cap. 1: Fundamentos

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

Bem vindo ao Google Colab!

Se você é novo no uso do Colab, assista os vários vídeos em português explicando a plataforma. Existem também **workbooks** introdutórios para o uso do Python e do Colab na página principal do projeto.

De forma muito resumida, o Colab combina **células de texto e de código**. Esse paradigma de juntar documentação e código em um único documento foi introduzido por Donald Knuth em 1984 [ref1, ref2].

1.1 Instruções

- É recomendado tirar uma cópia deste notebook clicando em **"Arquivo"**–>**"Salvar cópia no drive"**. Desta forma você poderá editá-lo e executar os campos de código sempre que quiser, sem perder os seu progresso ao fechar esta página. Essa cópia estará disponível na pasta **Colab Notebooks**, no seu Google Drive.
- Para poder editar uma **CÉLULA DE TEXTO** (como esta), basta dar dois cliques na célula e editar, que pode ser visualizada na aba à direita (ou abaixo), ou pressionar Shift+Enter, ou ainda clicando em outra célula.
- No Colab também tem a **CÉLULA DE CÓDIGO**, que pode ser executada com essa combinação de teclas, ou clique no botão **"executar"** ou **"play"** para ver a saída do código entrado.
- É recomendado seguir a ordem sugerida dos exercícios, uma vez que familiaridade com os conceitos introduzidos são esperados nos exercícios seguintes.
- É possível rodar códigos independentes em IDEs e também *online*, utilizando navegadores, como o Chrome: https://ideone.com/, https://www.codechef.com/ide,

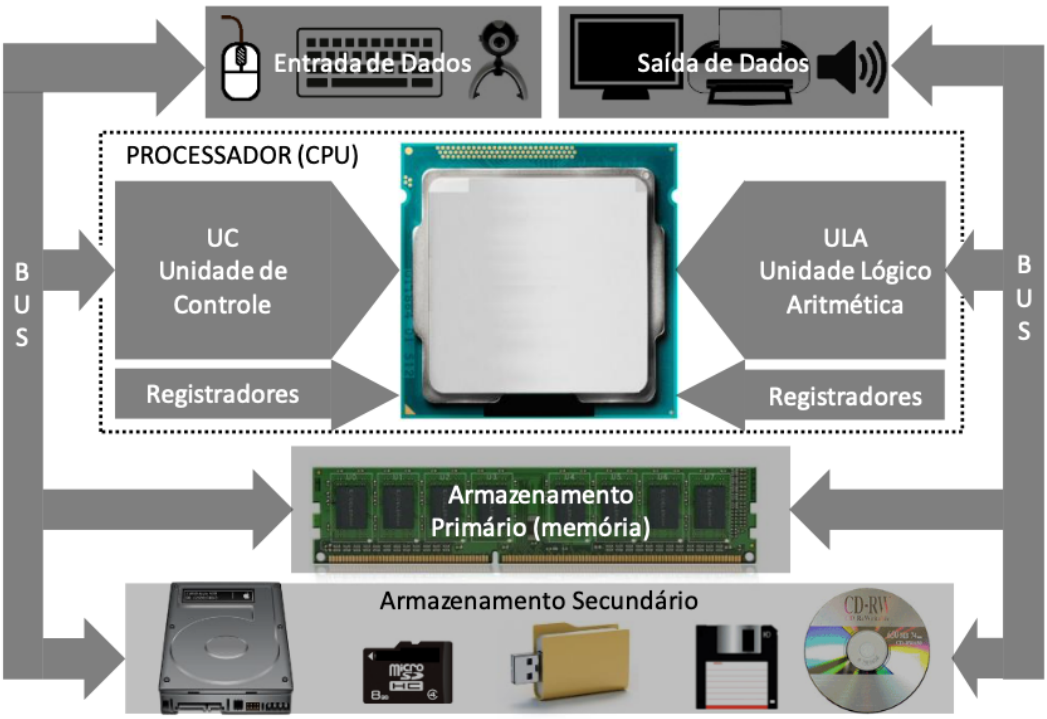
https://replit.com/, https://www.jdoodle.com
- É possível também rodar códigos no seu celular, utilizando navegadores acessando esses links.
- Esse arquivo com extensão `ipynb` pode ser editado também no Jupyter Notebook - https://jupyter.org.

1.2 Sumário

- Introdução à arquitetura de computadores; Hardware e Software
- Algoritmos, fluxogramas e lógica de programação
- Conceitos de linguagens de programação
- Variáveis, tipos de dados e organização da memória
- Operadores e precedência
- Aprendendo a programar
- Revisão deste capítulo
- Exercícios

1.3 Introdução à Arquitetura de Computadores

A arquitetura (ou organização dos principais componentes) mais conhecida de computadores foi introduzida por John Von Newmann, em 1936, ver Figura abaixo.



- Um software é um conjunto de linhas de código, armazenado em arquivo na memória secundária ou na RAM, que pode ser executado, instrução por instrução na CPU.
- Essas linhas de código são instruções que o computador consegue executar, mas que podem ter sido traduzidas de um **Algoritmo**.

1.4 Algoritmos, Fluxogramas e Lógica de Programação

```
ALGORITMO (algumas definições):
1. Um processo ou conjunto de regras a serem seguidas em cálculo ou outra
operação de solução de problemas, especialmente por um computador.
2. Processo de resolução de um problema constituído por uma sequência
ordenada e bem definida de passos que, em tempo finito, conduzem à
solução do problema ou indicam que, para o mesmo, não existem soluções.
```

Um exemplo de algoritmo: solução de uma equação do segundo grau no formato $ax^2 + bx + c$, dados a, b e c:

1. Calcule Δ com a fórmula $\Delta = b^2 - 4ac$;
2. Se $\Delta < 0$, x não possui raízes reais;
3. Se $\Delta = 0$, x possui duas raízes reais idênticas;
4. Se $\Delta > 0$, x possui duas raízes reais e distintas;
5. Calcule x usando a equação.

1.4.1 Pseudocódigo

- Essa sequência de passos definidas no algoritmo anterior, se incluída as instruções `leia(algo)` e `escreva(algo)`, poder ser definida como um **pseudocódigo**.
- Esse **pseudocódigo** é para humanos conseguirem entender os passos de um algoritmo, de uma forma "mais próxima" de como os computadores processam as instruções, utilizando uma linguagem de programação, definidas na próxima seção.

Veja a seguir um exemplo de pseudocódigo do algoritmo anterior:

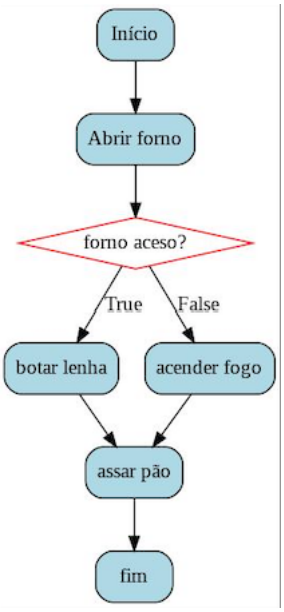
```
leia(a,b,c)
calcular delta = b**2-4*a*c
se delta < 0:
   escreva(x não possui raízes reais)
   fim do programa
se delta = 0:
   escreva(x possui duas raízes reais idênticas)
se delta > 0:
   escreva(x possui duas raízes reais e distintas)
calcule x=a*x*x+b*x+c
escreva(x)
```

Observe que em um pseudocódigo existe uma descrição um pouco mais detalhada das instruções, que em um algoritmo, mas não é uma descrição muito rígida/formal, como existem nas linguagems de programação.

1.4.2 Fluxogramas

- Os algoritmos, além de poderem ser representados por listas de instruções, como no último exemplo, podem ser representados graficamente para facilitar seu entendimento.
- Os fluxogramas e diagramas de atividades da UML (*Unified Modelling Language*) estão entre as representações mais usadas.

- Ambos, bem similares, usam formas e setas para indicar operações e seus fluxos.
- A Figura abaixo ilustra um exemplo de fluxograma.



Esse fluxograma foi gerado automaticamente rodando as duas células de código abaixo.

Observação: apesar do livro texto ser independente de linguagem, alguns códigos nesta adaptação em Colab serão apresentados na linguagem de programação Python, pela facilidade didática na apresentação de conteúdos. Assim, os detalhes desses códigos não serão apresentados, pois foge o escopo.

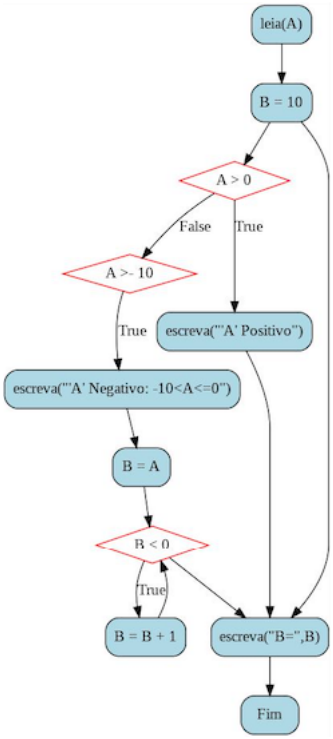
[ ]:
```
# instalando algumas bibliotecas no servidor do Colab (Linux)
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
# importando a biblioteca txtflow para gerar fluxograma a partir de␣
 ↪código
from txtoflow import txtoflow

# definindo o pseudocódigo e passando como entrada de dados para gerar␣
 ↪o fluxograma
txtoflow.generate(
    '''
    Início;
    Abrir forno;
    if (forno aceso?) {
        botar lenha;
    } else {
        acender fogo;
    }
    assar pão;
```

```
    fim;
    '''
)
```

- Após rodar a célula de código acima, ver o arquivo `flowchart.jpg` clicando no ícone de pasta à esquerda.



- Veja um outro fluxograma gerado a partir da célula abaixo; > Apesar deste primeiro fluxograma ser intuitivo, os detalhes (principalmente do próximo fluxograma) serão vistos durantes este curso!

[ ]:
```
# definindo uma variável do tipo texto (string) para armazenar o␣
 ↪pseudocódigo
alg2 = '''
leia(A);
B = 10;
if ( A > 0 ) {
  escreva("'A' Positivo");
} else if ( A >- 10 ) {
  escreva("'A' Negativo: -10<A<=0");
  B = A;
  while ( B < 0 ) {
    B = B + 1;
  }
}
escreva("B=",B);
Fim;'''

# criando o fluxograma
```

```
txtoflow.generate(alg2)
```

- Qual o valor final de B neste algoritmo, se o valor lido foi `A=-5`?
- Experimente também essa ferramenta *online*: code2flow

Fluxograma a partir de código python A sequência de códigos a seguir geram fluxogramas na própria célula de código, diferente de salvar em arquivo de imagem. Além disso, a entrada é um código em python, diferente do exemplo anterior que está em pseudocódigo.

[ ]:
```
!pip install --upgrade git+https://github.com/innovationOUtside/
↪flowchart_js_jp_proxy_widget.git
```

[ ]:
```
from google.colab import output
output.enable_custom_widget_manager()
```

[ ]:
```
alg3 = '''
A = int(input())
B = 10
if A > 0:
  print("'A' Positivo")
elif A > -10:
  print("'A' Negativo: -10<A<=0")
  B = A
  while B < 0:
    B = B + 1
    print(B)
print("B=",B)'''
```

[ ]:
```
from pyflowchart import Flowchart
fc = Flowchart.from_code(alg3)
flowchart = fc.flowchart()
```

[ ]:
```
from jp_flowchartjs.jp_flowchartjs import FlowchartWidget

testEmbed = FlowchartWidget()
testEmbed.charter(flowchart)
testEmbed
```

- Comparando os dois fluxogramas anteriores é possível notar que é mais difícil ler esse último por falta de cor e por ter cruzamentos entre retas.
- Achou alguma ferramenta mais interessante para gerar fluxograma neste Colab? Compartilhe!
  - Por exemplo, seria interessante criar um fluxograma a partir de uma célula de código, contendo várias instruções, e não a partir de um texto, como ocorre

na ferramenta *online* anterior.

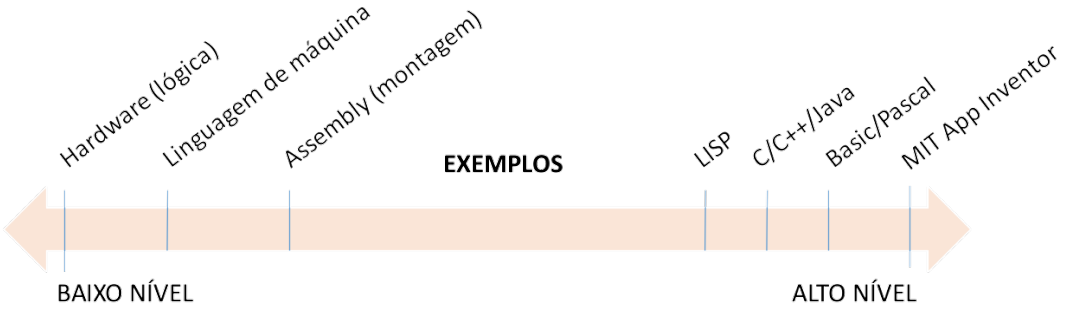
1.5 Conceitos de Linguagens de Programação

Linguagem é um conjunto de regras usado para se definir uma forma de comunicação. São conceitos fundamentais de qualquer linguagem, não só em computação:

- **sintaxe:** o arranjo de palavras e sua disposição em um discurso para criar frases bem formadas e relacionadas de forma lógica em uma linguagem;
- **semântica:** o ramo da linguística preocupado com a lógica e o significado das palavras. sub-ramos incluem a semântica formal e a semântica lexical.

1.5.1 Níveis de Liguagens

Nível de linguagem diz respeito à proximidade em que ela se encontra da linguagem natural humana.



1.5.2 Linguagem Compilada vs Interpretada

- A **compilação** é o nome dado ao processo de traduzir um código escrito em uma linguagem de mais alto-nível para código de máquina, que poderá ser executado na arquitetura apropriada para o qual foi compilado.
  - Exemplos de linguagens compiladas: C, CPP (ou C++), Pascal e Fortran.
- Em **linguagens interpretadas**, os comandos do programa são transformados em código nativo durante a sua execução, geralmente por um outro programa, necessário para seu funcionamento.
  - Exemplos de linguagens interpretadas: python (nesse Colab), R, matlab.
- A exceção é a linguagem **Java**, que compila um código binário em bytes (**bytecode**), para ser executado não em uma arquitetura real específica, mas sim pela **máquina virtual Java** (JVM), que é um **programa nativo** (isto é, é necessário um JVM compilado para cada arquitetura) responsável por transformar *bytecode* na linguagem de máquina nativa do processador em tempo de execução (**runtime**).
- A existência de JVM para a maioria das plataformas aumentou em muito a portabilidade dos programas. A **portabilidade** foi uma das principais motivações para a criação da linguagem Java.

1.5.3 Estruturas de Código

Independente da linguagem escolhida, as estruturas fundamentais de código que estarão presentes em todas elas são:

1. **Código sequencial:** os comandos são executados na ordem em que aparecem;
2. **Módulos de código (funções ou métodos):** conjuntos de comandos agrupados em sub-rotinas ou subprogramas;
3. **Desvios condicionais:** dada uma condição, existem dois caminhos possíveis (duas linhas) de execução;
4. **Estruturas de repetição ou laços:** conjuntos de comandos que se repetem enquanto uma dada condição for satisfeita.

1.5.4 Deburação de Código (DEBUG)

- É referido como depuração ou "debugação" o árduo processo de busca e correção dos erros de programação no código escrito em linguagens de programação.
- Ferramentas de depuração são de grande valia para auxiliar na busca e correção destes erros.
- Alguns dos erros comuns encontrados em código, do menos grave ao mais sério, são:
    1. **Erro de semântica:** palavras reservadas ou operações escritas de forma errada;
    2. **Erro de sintaxe:** envolve o uso inapropriado do formato dos comandos;
    3. **Erro de organização:** blocos de código, aspas ou uso de parênteses inconsistentes – ocorre quando se abre um bloco de código ou de operadores (com chaves, colchetes ou parênteses) ou um campo de texto (com aspas) sem fechar ou se fecha sem abrir;
    4. **Erro de lógica:** também conhecido como *Cerberus* (o cão de guarda do inferno), ocorre quando os comandos estão sintaticamente corretos, na semântica correta, e escritos consistentemente, porém em ordem, forma ou disposição que produz um resultado diferente do desejado. Resultado este que, frequentemente, causa travamento da execução do programa ou até mesmo do computador.

1.5.5 Ambientes de Desenvolvimento Integrados

Ambientes de desenvolvimento integrado ou IDE (*Integrated Developing Environment*), como são conhecidos, oferecem uma combinação de ferramentas úteis para escrita, execução e depuração do código de programas.

Algumas ferramentas de desenvolvimento populares do tipo IDE são: - **Eclipse** (C, CPP, Java, PHP, android, etc); - **NetBeans** (C, CPP, Java, ruby, HTMl5, PHP, etc); - **PyCharm** (Python, JavaScript, CoffeeScript, XML, HTML/XHTML); - **Xcode** (IDE de desenvolvimento para dispositivos Apple); - **Visual studio** (C, CPP, C#, visual Basic, entre outros, voltado para o desenvolvimento de programas na plataforma Windows). - **VSCODE** (versão *online* de código aberto do *Visual studio*. Para download e uso local: https://code.visualstudio.com/download).

Mais alguns links úteis para programar em várias linguagens *online*:

- https://colab.research.google.com
- https://replit.com/languages
- http://www.drjava.org
- https://sourceforge.net/projects/nbportable
- https://ideone.com
- https://www.codechef.com/ide

  Python:

- https://www.programiz.com/python-programming/online-compiler/
- https://www.onlinegdb.com/online_python_compiler
- https://www.tutorialspoint.com/execute_python_online.php
- https://www.python.org/shell/
- http://pythontutor.com

Ou para Baixar IDE's: * https://www.anaconda.com/distribution * https://netbeans.org

Mais links interessantes: * https://learnxinyminutes.com/docs/python3 * https://learnxinyminutes.com/docs/java * https://learnxinyminutes.com/docs/c * https://youtu.be/UNSoPa-XQN0

Tem outros links interessantes que não estão nestas listas? Compartilhe!

Algumas Estatísticas sobre Linguagens Mais Usadas:

- https://insights.stackoverflow.com/trends?tags=python%2Cjava%2Cjavascript

- reportagem
- https://insights.stackoverflow.com/survey/2020
- https://insights.stackoverflow.com/survey/2020#most-popular-technologies

1.6 Variáveis

- Uma variável em programação de computadores é um indicador ou 'apelido' (*alias*) atribuído a um endereço de memória que contém um dado guardado.
- Os elementos na memória, representados por variáveis, podem conter apenas um ou múltiplos valores, e são codificados na memória de acordo com o tipo específico do dado, com formato e número de bits apropriado para mantê-los.

1.6.1 Uso de Variáveis

Em qualquer linguagem, um valor pode ser atribuído a uma variável através do operador de atribuição, por exemplo: "=".

Exemplos de variáveis do tipo inteiro:

- As próximas células de código em Python podem ser executadas clicando em `[ ]` ou seta de *play*, na margem esquerda.
- Se o seu objetivo é aprender outras linguagens de programação, essas células em Python podem ser utilizadas como comparativos.

```
[ ]: idade = 20;      # esse é um comentário
     anoAtual = 2021 # observer que ';' é opcional em python e javascript
```

A leitura correta para o código representado acima é **"a variável `idade` recebe o valor 20"**.

```
[ ]: idade
```

```
[ ]: anoNascimento = anoAtual - idade
     anoNascimento
```

Exemplo de variável do tipo texto (*string*)

```
[ ]: nomeAluno = 'Rafael Morais'
     nomeAluno
```

```
[ ]: nomeAluno = "Rafael Morais"
     nomeAluno
```

Observe que o texto deve ficar entre aspas simples ou duplas (dependendo das linguagem escolhida).

Exemplo de variável do tipo real

```
[ ]: pi = 3.1415926535897932
     pi
```

Observe que tem que usar '.' para separar as casas decimais.

Nomenclatura

- Os nomes das variáveis podem conter letras, números e alguns caracteres especiais, desde que não sejam empregados pelo lexema da linguagem (símbolos dos operadores matemáticos, lógicos e relacionais, por exemplo).
- Em geral, o nome da variável deve começar por uma letra ou, dependendo da linguagem, algum caractere especial.
- Como regra geral os nomes de variáveis não devem começar com números ou conter espaços.

Exemplos de variáveis válidas e inválidas:

Válido	Inválido
idade4	4idade
Nome_Completo	Nome Completo
resta_um	resta-um
V8S9F0D7S7D9	V@R!V&+

1.7 Operadores e precedência

As linguagens reservam alguns caracteres para uso em operadores ou indicadores de tipo.

1.7.1 Operadores aritmeticos

**

Tabela: Operadores aritméticos em Java/C/CPP/JS

**

operador	descrição
+	soma com
-	subtração por
*	multiplicação por
/	divisão por
%	resto da divisão por
++	incremento
–	decremento

1.7.2 Operadores relacionais

**

Tabela: Operadores relacionais em Java/C/CPP/JS

**

operador	descrição
==	é igual a
!=	é diferente de
>	é maior que
<	é menor que
>=	é maior ou igual a
<=	é menor ou igual a

1.7.3 Operadores lógicos

**

Tabela: Operadores lógicos binários em Java/C/CPP/JS

**

operador	descrição
x && y	True se ambos também forem
x & y	True se ambos também forem bit-a-bit
x || y	False se ambos também forem
x | y	False se ambos também forem bit-a-bit
! x	contrário de x
~ x	contrário de x bit-a-bit
^ x	ou exclusivo `XOR` bit-a-bit
«N	*shift-left*, adiciona N zeros à direita do número binário
«N	*shift-right*, elimina N zeros à direita do número binário

1.7.4 Operadores de atribuição

**

Tabela: Operadores de atribuição em Python/Java/C/CPP/JS

**

operador	descrição
=	recebe o valor de
+=	é somado ao valor de
-=	é subtraído do valor de
*=	é multiplicado pelo valor de
/=	é dividido pelo valor de
%=	recebe o resto da divisão por
similarmente	»=, «=, &=,

1.7.5 Precedência de Operadores

Quando utilizados em uma expressão, os operadores apresentados anteriormente são executados em ordem de precedência, de forma semelhante como fazemos em equações matemáticas, com somas, subtrações, multiplicações, divisões, exponenciais, etc. (testar Tabela abaixo na linguagem escolhida).

**

Tabela: Precedência de Operadores

**

Categoria	Operador	Associatividade
Pós-fixado	() (parênteses, operador ponto)	→ Esquerda para a direita

Categoria	Operador	Associatividade
Índice	[]	→ Esquerda para a direita
Unário	++ - - ! ~ (incremento, decremento)	← Direita para a esquerda
Multiplicativo	* / % (vezes, dividido, resto)	→ Esquerda para a direita
Aditivo	+ - (soma, subtração)	→ Esquerda para a direita
Shift binário	» »> «	→ Esquerda para a direita
Relacional	> >= < <=	→ Esquerda para a direita
Igualdade	== != (é igual a, é diferente de)	→ Esquerda para a direita
E (AND) bit-a-bit	&	→ Esquerda para a direita
XOR bit-a-bit	^	→ Esquerda para a direita
Ou (OR) bit-a-bit	|	→ Esquerda para a direita
E lógico	&&	→ Esquerda para a direita
Ou lógico	||	→ Esquerda para a direita
Condicional	?:	← Direita para a esquerda
Atribuição	= += -= *= /= %= »= «= &= ^= |=	← Direita para a esquerda

Tipos básicos de variáveis

Conversões de tipos de variáveis **

Tabela: Tipos de variáveis em Java/C/CPP/JS

**

Tipo	**Descrição**	**n. Bits**	**Mínimo**	**Máximo (valor)**
Tipos inteiros+sinal				
byte	inteiro de 1 byte	8	-128	127
short	inteiro curto	16	-2^{15}	$2^{15}-1$
int	inteiro	32	-2^{31}	$2^{31}-1$
long	inteiro longo	64	-2^{63}	$2^{63}-1$

Tipos reais+ponto flut.				
float	precisão simples	32	2^{-149}	$(2-2^{-23})2^{127}$
double	precisão dupla	64	2^{-1074}	$(2-2^{-52}))2^{1023}$
Tipos lógicos				
boolean	valor booleano	1	$false$	$true$
Tipos alfanuméricos				
char	caractere unicode	16	0	$2^{16}-1$
Classecadeia de caract.				
String	sequência n chars	$16*n$	-	-
Outras				
void	variável vazia	0	-	-

**

Tabela: Exemplos de conversão de valores entre tipos de variáveis

**

Conversão	**Método**	**Exemplo**
		C/C++/Java
float → int	Type cast	`int i = (int) varFloat;`
double → float	Type cast	`float f = (float) varDouble;`
float, int → double	Direto	`double d = i; d = f;`
Número → String	(Java) direto (C) stdlib.h	`String str = + f; str = snprintf(num);`
String → int	(Java) parse (C) stdlib.h	`i = Integer.parseInt(str); i = atoi(str);`
String → float	(Java) parse (C) stdlib.h	`f = Float.parseFloat(str);f = atof(str);`
String → double	(Java) parse (C) stdlib.h	`d = Double.parseDouble(str);d = strtod(str, NULL);`
		JavaScript
String → int	Parse	`var i = parseInt(str);`
String → float	Parse	`var f = parseFloat(str);`

Número → String	toString()	`var str = toString(f);`

1.8 Teste de mesa

- Teste de mesa é o nome dado à simulação manual da execução de um programa, acompanhando o estado das variáveis e a mudança temporal de seus valores, quando feito no papel ou mesmo mentalmente.
- Geralmente anota-se o nome das variáveis, a medida em que aparecem, e seus respectivos valores. Quando as variáveis são modificadas, os novos valores vão substituindo os anteriores, que são atualizados (riscados) na tabela para cada nova instrução que as modifica.
- Os valores finais das variáveis, ao término do programa, são os últimos valores assumidos por cada uma delas.

Veja um exemplo de teste de mesa, apresentado a seguir:

código	c	f	ano	idd
`c = 1`	1			
`f = 22`		22		
`ano = 1994`			1994	
`idd = 0`				0
`ano=ano+idd`			1994	
`idd *= f`				0
`c += 1`	2			
`ano+=f`			2016	
`f -= 4`		18		
`c += 1`	3			

1.9 Aprendendo a programar

- Uma sugestão prática para experimentar algumas linguagens de programação é começar escrevendo um programa bem simples.
- Escreve um programa para imprimir a mensagem:

```
Alô, Mundo!
```

Salvando um arquivo Para salvar um arquivo contendo os códigos de uma célula de código, basta colar na primeira linha o comando `%%writefile nomeArquivo.ext`.

Exemplo 01 - Escreva 'Alô Mundo!'

Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto:

```
case=caso1
output=Alô, Mundo!
```

- Então, quando o estudante submeter um código em uma atividade VPL no Moodle, o servidor de correções irá comparar entradas e saídas apenas!
- Ou seja, o código submetido pelo estudande deve **ler as entradas (se existirem) e gerar a saída esperada, independente de linguagem de programação**.

[1]:
```
%%writefile cap1ex01.c
#include <stdio.h>
int main(void) {
  printf("Alô, Mundo!");
  return 0;
}
```

[ ]:
```
!gcc -Wall -std=c99 cap1ex01.c -o output2
!./output2
```

- O comando `%%writefile` salva o arquivo no servidor *colab* (pasta à esquerda).
- O comando após `!` roda no terminal.
- `gcc` é o compilador utilizado, com os argumentos:
  - `-Wall` para mostrar *warnings* (programas mal feitos)
  - `-std=c99` compilador padrão ANSI (`c99` é um padrão de 1999, porém existem outros, ver por exemplo ref1 e ref2), compatível em mais arquiteturas
  - arquivo de entrada `cap1ex01.c`
  - `-o` arquivo de saída executável

É possível copiar e colar o código acima (sem a primeira linha iniciada com `%%writefile arquivo.ext`) e rodar localmente como no Terminal (*Shell* ou *Console*), em IDEs, ou online. Experimente!

Após rodar a célula de código acima, ver esse arquivo clicando no ícone de pasta à esquerda.

Clicar no arquivo, depois nos três pontinhos e fazer download para o seu computador. É possível editar e executar esse arquivos em IDE's instaladas no seu computador (apps no seu celular), ou também de *online*.

O Colab tem a linguagem Python nativa nas células de código.

1.9.1 Programação sequencial

Programação incorpora conceitos de matemática e de lógica, entre eles, variáveis e expressões algébricas. Como na matemática, a expressão a seguir produzirá C = 10.

```
int A = 2, B = 3;
int C = (A+B) * 2;
```

1.9.2 Entrada de dados

Exemplo(s) 02 - Entrada de Dados

Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 02, basta incluir em `Casos para teste`, o seguinte texto, com 4 casos (pode incluir mais casos):

```
case=caso1
input=65
output=
65 graus Celsius corresponde a 149.0 graus Fahrenheit
case=caso2
input=55
output=
55 graus Celsius corresponde a 131.0 graus Fahrenheit
case=caso3
input=45
output=
45 graus Celsius corresponde a 113.0 graus Fahrenheit
case=caso4
input=35
output=
35 graus Celsius corresponde a 95.0 graus Fahrenheit
```

[2]:
```
%%writefile cap1ex02.c
#include <stdio.h>
int main(void) {
  int C;
  scanf("%d", &C);
  float F = C * 9/5 + 32;
  printf("%d graus Celsius corresponde a %.1f graus Fahrenheit", C, F);
  return 0;
}
```

[3]:
```
%%shell
gcc -Wall -std=c99 cap1ex02.c -o output2
./output2
```

1.9.3 Divisão de um código em três partes

- O uso eficaz de cada linguagem depende apenas da familiaridade do programador com a sintaxe, semântica e bibliotecas disponíveis, o que só pode ser atingido com a prática.
- Adicionalmente, para se incorporar *inteligência* aos programas (ou boas práticas de programação), é necessário:
  - conhecimento de lógica de programação para saber como estruturar e
  - organizar os programas de forma a criar um fluxo contínuo de código:

* partindo da **ENTRADA** (ou coleta) de dados,
* seguido pelo **PROCESSAMENTO** da informação,
* até a **SAÍDA** de dados, com a exibição dos resultados do processamento para o usuário.

ENTRADA DE DADOS ⇒ PROCESSAMENTO DA INFORMAÇÃO ⇒ SAÍDA

As estruturas fundamentais de lógica de programação, usadas para orientar o fluxo do processamento, comuns a todas as linguagens de programação, são apresentadas nos próximos três capítulos.

1.9.4 Tipos de dados em C

O comando sizeof retorna a quantidade de bytes de cada variável.

[ ]:
```
%%writefile cap1ex03.c
#include <stdio.h>
int main(void) {
  char ch;
  int i;
  long int li;
  short s;
  unsigned int ui;
  float f;
  double d;
  long double ld;

  printf("Numero de bytes por tipo de dados:\n");
  printf("char:          %ld\n", sizeof (ch));
  printf("int:           %ld\n", sizeof (i));
  printf("long int:      %ld\n", sizeof (li));
  printf("short:         %ld\n", sizeof (s));
  printf("unsigned int:  %ld\n", sizeof (ui));
  printf("float:         %ld\n", sizeof (f));
  printf("double:        %ld\n", sizeof (d));
  printf("long double:   %ld\n", sizeof (ld));
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap1ex03.c -std=c99 -Wall -o output3
./output3
```

1.9.5 Casts

Para alterar um tipo de dados para outro utilizar *cast*.

[ ]:
```
%%writefile cap1ex04.c
#include <stdio.h>
int main(void) {
  double d = 3.14;
  int i = (int)d; // AQUI ESTA O CAST
  printf("double = %f\n", d);
  printf("int = %d\n", i);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap1ex04.c -std=c99 -Wall -o output4
./output4
```

1.10 Exercícios

Ver notebook Colab no arquivo `cap1.part2.lab.*.ipynb` (`*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, onganizadas em subpastas contidas de `"gen"`, na pasta do Google Drive colabs.

1.11 Revisão deste capítulo de Fundamentos

- Introdução à arquitetura de computadores; Hardware e Software
  - Destaque para a arquitetura de John Von Newmann
- Algoritmos, fluxogramas e lógica de programação
  - Algoritmo define um conjunto de passos para o computador executar
- Conceitos de linguagens de programação
  - Linguagens Compiladas vs Interpertadas
- Variáveis, tipos de dados e organização da memória
  - Utilizar nomes sugestivos para as variáveis e definir corretamente o seu tipo
- Operadores e precedência
  - Praticar a precedência de operadores em expressões matemáticas na linguagem escolhida
- Aprendendo a programar
  - **Pratique! Só se aprende a programar se fizer todos os exercícios, sem copiar soluções prontas!**
- Exercícios

1.12 Processando a Informação: Cap. 1: Fundamentos - Prática 1

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

1.12.1 Instruções

- A lista de exercícios PODERÁ NÃO SER AVALIADA PELO PROFESSOR, mas entre em contato com o monitor ou o professor se tiver alguma dúvida.
- É recomendado tirar uma cópia do notebook clicando na opção **"copiar para o drive"** no menu acima à direita (você precisa estar logado na sua conta no Google para fazer isto). Alternativamente, selecione **"Arquivo"**–>**"Salvar cópia no drive"**. Desta forma você poderá editá-lo e executar os campos de código sempre que quiser, sem perder os seu progresso ao fechar esta página.
- Após digitar o código pedido no campo indicado, clique no botão **"executar"** ou **"play"** para ver a saída do código entrado.
- É recomendado seguir a ordem sugerida dos exercícios, uma vez que familiaridade com os conceitos introduzidos são esperados nos exercícios seguintes.
- O *notebook Colab* tem o Python nativo para rodar as células de código. Se desejar outra linguagem de programação, consulte o material do Capítulo 1, na pasta do Google Drive colabs

1.12.2 Sugestões para envio dos exercícios com avaliação automática no Moodle+VPL

- Fique atento as datas de entrega no calendário do Moodle.
- Escolha uma linguagem de programação para submeter os exercícios (ou a definida pelo seu professor).
- A atividade no Moodle considera sempre a última submissão, com a nota atribuída de forma automática.

- Faça uma cópia do Colab e tente resolver as questões aqui, ou em alguma IDE de sua preferência, antes de enviar para avaliação no Moodle.
- É possível rodar códigos independentes em IDEs e também *online*, utilizando navegadores como o chrome: https://ideone.com/, https://www.codechef.com/ide, https://replit.com/, https://www.jdoodle.com.
- É possível também rodar códigos no seu celular, utilizando navegadores acessando esses links.

1.12.3 Exercícios

---

1. No espaço abaixo, escreva um programa para exibir a mensagem "Alô mundo!". Execute-o e verifique se a saída está correta.

---

2. **Complete** o processamento do código abaixo com a fórmula correta para o delta ($\Delta = b^2 - 4ac$) da equação do segundo grau, na forma $ax^2 + bx + c = 0$. **Dados**: a operação 'elevado' em Python é `**`. Considere alguns valores inteiros e reais para a, b e c. **Resposta correta:** delta = 49

```
a = 2
b = 3
c = -5
```

---

3. **Complete o código anterior** no campo abaixo com comandos para calcular e exibir as raízes da equação do segundo grau, usando a fórmula $raiz_1 = \frac{-b+\sqrt{\Delta}}{2a}$ e $raiz_2 = \frac{-b-\sqrt{\Delta}}{2a}$. **Dados**: raiz de um número = número**.5 (elevado a meio). **Respostas:** (-2.5, 1.0).

---

4. Adicione `código` na parte indicada para trocar os valores das variáveis `a, b e c` (definidas anteriormente) de forma que `a` contenha o menor valor, `b` contenha o valor central e `c` contenha o maior valor. A última linha `(a, b, c)` não deve ser alterada. **Saída**: `(-5, 2, 3)`

---

5. Escreva abaixo um programa completo que leia dois números (devem ser digitados pelo usuário) e calcule e exiba a média. Quando executado, o programa deve gerar a seguinte saída:

```
Entre com o primeiro número: 10
Entre com o segundo número: 5
Média: 7.5
```

1.13 Processando a Informação: Cap. 1: Fundamentos - Prática 2

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

1.13.1 Exercícios

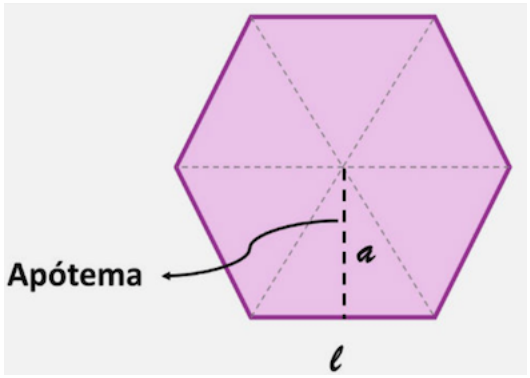
[Fonte: https://wiki.python.org.br/EstruturaSequencial]

---

1. Faça um programa que peça o raio (R) de um círculo, calcule e mostre sua área (πR^2) e seu perímetro ($2\pi R$).

---

2. Faça um programa que peça a Apótema (a, ver figura), o número (n) de lados do polígono regular e o comprimento l de cada lado. Calcule e mostre a sua Área ($A = nla/2$). Essa fórmula funciona para triângulo equilátero e quadrado? [fonte]



---

3. Faça um programa que pergunte quanto você ganha por hora e o número de horas trabalhadas por dia da semana (sem sábados e domingos). Calcule e mostre o total do seu salário semanal.

---

4. Faça um programa que peça a temperatura em graus Fahrenheit, transforme e mostre a temperatura em graus Celsius.

---

5. Faça um programa que peça 2 números inteiros e um número real. Calcule e mostre:

- o produto do dobro do primeiro com metade do segundo;
- a soma do triplo do primeiro com o terceiro;
- o terceiro elevado ao cubo.

---

6. Tendo como dados de entrada a altura (h) de uma pessoa, construa um programa que calcule seu peso ideal, usando a seguinte fórmula:

$$72.7h - 58$$

---

7. Tendo como dado de entrada a altura (h) de uma pessoa, construa um programa que calcule seu peso ideal, utilizando as seguintes fórmulas:

- homens: $72.7h - 58$
- mulheres: $62.1h - 44.7$

---

8. João Papo-de-Pescador, homem de bem, comprou um microcomputador para controlar o rendimento diário de seu trabalho. Considere que ele sempre traz um peso de peixes maior que o estabelecido pelo regulamento de pesca do estado de São Paulo (50 quilos), neste caso, com uma multa de R$ 4,00 por quilo excedente. João precisa que você faça um programa que leia a variável peso (peso de peixes) e calcule o excesso. Gravar na variável excesso a quantidade de quilos além do limite e na variável multa o valor da multa que João deverá pagar. Imprima os dados do programa com as mensagens adequadas.

---

9. Faça um programa que pergunte quanto você ganha por hora e o número de horas trabalhadas no mês. Calcule e mostre o total do seu salário no referido mês, sabendo-se que são descontados 11% para o Imposto de Renda, 8% para o INSS e 5% para o sindicato, faça um programa que nos dê:

- salário bruto.
- quanto pagou ao INSS.
- quanto pagou ao sindicato.
- o salário líquido.
- calcule os descontos e o salário líquido, conforme a tabela abaixo:

```
+ Salário Bruto : R$
- IR (11%) : R$
- INSS (8%) : R$
- Sindicato  (5%) : R$
= Salário Liquido : R$
```

Obs.: Salário Bruto - Descontos = Salário Líquido

---

10. Faça um programa para uma loja de tintas. O programa deverá pedir o tamanho em metros quadrados da área a ser pintada. Considere que a cobertura da tinta é de 1 litro para cada 3 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam R$ 80,00. Informe ao usuário a quantidade de latas de tinta a serem compradas e o preço total.

1.14 Processando a Informação: Cap. 1: Fundamentos - Prática 4: Tutorial VSCode

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

1.14.1 Tuturial (em construção) VSCode

- **VSCODE** - versão *online* de código aberto do *Visual studio*.
- Para download e uso local: https://code.visualstudio.com/download.
- Ver:
  - https://docs.microsoft.com/pt-br/learn/modules/introduction-to-github-visual-studio-code/

  - https://www.digitalocean.com/community/tutorials/how-to-use-git-integration-in-visual-studio-code-pt
  - https://balta.io/blog/git-github-primeiros-passos
- Clonar: https://github.com/fzampirolli/codigosPE
- Alterar `settings.json` em `code-runner.executorMap`:
  - `"c": "cd $dir && gcc -Wall -std=c99 $fileName -o ...`
    * `-Wall` para mostrar `warning` (programas ruins)
    * `-std=c99` - padrão ANSI ou C99, os programas seguindo esse padrão são mais portáveis, ou seja, possuem menos problemas de compatibilidade entre diferentes arquiteturas.

2 Processando a Informação: Cap. 2: Organização de código

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

2.1 Sumário

- Revisão do capítulo anterior
- Programas sequenciais
- Comentários
- Desvios de fluxo
- Programas e Subprogramas
- Funções, métodos e modularização
- Reaproveitamento e manutenção de código
- Revisão deste capítulo
- Exercícios

2.2 Revisão do capítulo anterior (Fundamentos)

- No capítulo anterior foram apresentados os fundamentos para se iniciar a programar utilizando alguma linguagem de programação, além de alguns exemplos de códigos e principalmente de como executar estes códigos em ambientes de programação.
- Neste capítulo iremos iniciar a organizar esses códigos, utilizando o conceito de sistema de informação em partes:
  - **ENTRADA DE DADOS ⇒ PROCESSAMENTO DA INFORMAÇÃO ⇒ SAÍDA**

2.3 Programas sequenciais

- Como vimos no capítulo anterior, um programa consiste de uma **sequência lógica de comandos e operações** que, executadas em ordem, realizam uma determinada tarefa.
- Em geral, um programa processa dados de entrada de forma a obter o resultado desejado ou saída.
- A **entrada** e a **saída** de dados são, comumente, realizadas utilizando os dispositivos de entrada e saída, como teclado e monitor.
- Os dados requeridos para o **processamento** podem estar também contidos no código do programa (no exemplo a seguir, poderia ser `conta = 100`).
- No exemplo a seguir em **pseudocódigo** a entrada é lida do teclado (comando `leia()`) e a saída é impressa no monitor (comando `escreva`).

```
#1 ENTRADA de dados:
#1.1 Definição das variáveis do programa:
Real: conta, gorjeta;

#1.2 Entrada de dados
conta = leia("Entre o valor da conta: ");

#2 PROCESSAMENTO:
gorjeta = conta * 0.2;

#3 SAÍDA de dados (na tela):
escreva("Valor da gorjeta = " + gorjeta);
```

Assim, o procedimento para especificar um programa é definir: 1. Os dados de **entrada** necessários (insumos) 2. O **processamento** ou transformação dos dados de entrada em saída 3. A **saída** desejada do programa

2.4 Comentários

- Comentários são partes do código usados pelo desenvolvedor para deixar notas, explicações, exemplos, etc.
- Quando definido um comentário, é dada uma instrução direta ao **compilador/interpretador para ignorar a parte comentada**.
- Isto quer dizer que os comentários não serão considerados quando o código for executado.
- Logo, os comentários não são parte da execução.
- Cada linguagem tem sua própria maneira de introduzir comentários no código.

Tabela. Identificadores de comentário.

Linguagem	Uma linha	Várias linhas
Java/JS/C/C++	//...	/*...*/
Python	#...	“““...”“” ou ’’’...’’’
Matlab	%...	%{...}%

Tabela. Identificadores de comentário.		
Pascal	{...}	{...}
SQL	–...	/.../
R	#..	

2.5 Desvio de Fluxo

- Um programa consiste em uma sequência de comandos executados em ordem, em uma linha contínua de execução.
- No entanto, esta linha (em inglês: *thread*) pode conter desvios ou descontinuidades, processando códigos de bibliotecas ou subprogramas.

2.6 Programas e Subprogramas

- Em grande parte das linguagens de programação, **o código dos programas pode ser dividido em um ou mais arquivos ou partes**.
- Cada parte contem uma sequência de comandos com um objetivo, realizando uma tarefa dentro do programa.
- Dentro de um mesmo programa podem existir **subprogramas** (ou partes) com funções específicas ou subconjuntos de comandos que só serão executados em condições especiais.
- Todas as linguagens vêm acompanhadas de **bibliotecas**, estas contendo funções ou programas de uso comum.
- São exemplos as funções para cálculos matemáticos, para operações de entrada e saída, para comunicação e conversão de dados.

2.7 Bibliotecas

Cada linguagem de programação possui um conjunto de bibliotecas disponíveis para uso. As bibliotecas podem guardar variáveis ou funções.

2.8 Funções ou Métodos de Usuário

- O uso de funções facilita a **reutilização de código**, dado que uma função é um programa autocontido, com **entrada**, **processamento** e **saída**.
- Uma função pode ser copiada de um programa para outro ou incorporado em uma biblioteca escrita pelo usuário, utilizando o comando em Python `import myBiblioteca`. Ou também, para C/C++/Java, `#include "myBiblioteca.h"`.
- Uma função é definida por um nome, retorno (opcional), argumento(s) (opcional) e um conjunto de instruções.
- A seguir temos um exemplo de função de usuário escrita em pseudocódigo:

```
# MINHA(S) FUNÇÃO(ÕES)
função delta(recebe: real a, real b, real c) retorna real d {
```

```
    d = b2 - 4ac
    retorne d
}

principal {
  # ENTRADAS
  a = 5
  b = -2
  c = 4

  # PROCESSAMENTO
  real valor = delta(a, b, c) # AQUI ESTÁ A CHAMADA DA FUNÇÃO

  # SAÍDA
  escreva(“O delta de ax2 +bx + c é ” + valor)
}
```

```
TERMINOLOGIA: Os métodos podem ser chamados também de
módulos, funções, subprogramas ou procedimentos.

Existe uma convenção: quando um método tem argumento(s)
(parâmetros) e um retorno é chamado de função, caso contrário,
é chamado procedimento.

Porém, poucas linguagens fazem distinção na sintaxe entre
função e procedimento, como a Pascal, tornando confusa
esta convenção.
```

Exemplo 01 - Uso de Funções Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
output=
O delta de ax^2 + bx + c é -76.0
```

- Quando uma função não tem `return` ela deve retornar o tipo `void` (nada), por exemplo, `void escrevaDelta(float d){...}`.
- Os argumentos de uma função podem ser passados por **valor** ou por **referência**.
- Por **valor**: qualquer alteração do argumento dentro da função não será passada para quem chamou a função.
- Por **referência**: oposto ao anterior, qualquer alteração dentro da função será repassada para quem chamou e isso ocorre passando o endereço de memória da variável (com `&`) ao ser chamado. Por exemplo: * `incrementaUm(&x)`
- Esse endereço de memória é definido como o tipo de dados **ponteiro** e será estudado em detalhes na parte de alocação estática vs dinâmicas.

- Para acessar o conteúdo de uma variável ponteiro, usar o prefixo `*`. Por exemplo:
  - ```
    void incrementaUm(int *x) {
        *x=*x+1;
    }
    ```
- Observe que o **ESCOPO** da função é definido por `{ ... }`.
- Experimente também essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[ ]:
```
%%writefile cap2ex01.c
#include <stdio.h>
// MÉTODO

float delta(float a, float b, float c) {
  float d = b * b - 4 * a * c;
  return d;
}
// PROGRAMA PRINCIPAL
int main(void) {
  float a = 5.0, b = -2.0, c = 4.0;
  printf("O delta de ax^2 + bx + c e %.1f\n", delta(a, b, c));
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap2ex01.c -o output2
./output2
```

2.8.1 Tabulação

- Um conceito muito importante em programação é a "endentação" (*indentation*) ou tabulação do código.
- Note que sempre que um bloco ou subconjunto de comandos é iniciado com `{` a tabulação é incrementada, e quando um subconjunto de comandos se encerra com `}` a tabulação é recuada.
- Isto permite visualizar claramente quando um grupo de comandos define um subprograma.
- Alguns editores de programa e a maioria das IDEs já fazem a tabulação de forma automática. Pesquise como fazer isso em uma IDE que esteja utilizando.
- Códigos sem tabulações corretas são muito difíceis de ler por outra pessoa (**os professores geralmente descontam pontos em códigos desorganizados!**).

2.9 Escopo

- Os subprogramas são programas independentes dentro do programa.
```

- Logo possuem variáveis próprias para armazenar seus dados.
- Estas **variáveis locais** só existem no âmbito do subprograma e só durante cada execução (chamada) do mesmo, desaparecendo (ou sendo apagadas) ao término do subprograma ou ao retornar em qualquer ponto com o comando `return`.
- **Variáveis globais**, por outro lado, podem ser criadas em um escopo hierarquicamente superior a todos os métodos/funções, desta forma permeando todos os subprogramas.
- Logo, as **variáveis globais** têm escopo em todos os métodos.

2.10 Reaproveitamento e Manutenção de Código

- Outra vantagem do uso de funções/métodos, além da capacidade de se reaproveitar o código já escrito em novos programas copiando os subprogramas desejados, é
  - a possibilidade de se atualizar os métodos sem a necessidade de alterar o código do programa principal.
- Para tanto, basta que a comunicação do método (entradas e saídas) permaneça inalterada.
- Como exemplo, utilizamos um programa com métodos para entrada e saída de dados com os métodos/funções `leia()` e `escreva()`, baseado nos exemplos anteriores.
- Para programas muito simples, como poucas linhas de código, pode ter a impressão de deixar o código mais complicado, mas a principal vantagem é o reaproveitamento de código em outros programas similares.
- Esse recurso de métodos de entrada e saída serão muito úteis nos tópicos de Vetores e Matrízes, abordados nos Capítulos 5 e 6, respectivamente,
  - quando métodos para ler e escrever um vetor/matriz poderão ser reaproveitados em várias questões.
- Para não ter muitas cópias desses métodos, é possível criar as nossas bibliotecas.

Para criar uma biblioteca em C devemos:

1. Criar um arquivo *header*, por exemplo `meusMetodos.h`, contendo as assinaturas dos métodos. Para o Exemplo 2 a seguir, esse arquivo deve conter:

```
float delta(float a, float b, float c);
float leia();
```

2. Criar um arquivo com as implementações dos métodos, incluindo no início `#include "meusMetodos.h"`.
   - Observar que não devemos utilizar `<>`, como ocorrem nas bibliotecas padrão do C, como exemplo: `#include <stdio.h>`.
3. Nos programas que irão utilizar esses métodos, incluir também no início a biblioteca criada, por exemplo: `#include "meusMetodos.h"`.
4. Para compilar e rodar, basta fazer:

```
gcc -Wall -std=c99 meusMetodos.c meusProgramas.c -o meusProgramas.o
./meusProgramas.o
```

Exemplo 02 - Uso de Funções com Entrada e Saída de Dados Casos para Teste no Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 02, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=3
4
5
output=
-44.0
case=caso2
input=3
4
2
output=
-8.0
case=caso3
input=3
5
2
output=
1.0
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[ ]:
```
%%writefile cap2ex02.c
#include <stdio.h>

float delta(float a, float b, float c) {
  float d = b*b-4*a*c;
  return d;
}

float leia() {
  float valor;
  printf("Entre com um valor: ");
  scanf("%f", &valor);
  return valor;
}

int main(void) {

  // ENTRADAS
```

```
  float a, b, c, d;
  a = leia();
  b = leia();
  c = leia();

  // PROCESSAMENTO
  d = delta(a, b, c);

  //SAÍDA
  printf("Delta = %.1f", d);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap2ex02.c -o output2
./output2
```

2.11 Exercícios

Ver notebook Colab no arquivo `cap2.part2.lab.*.ipynb` (`*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, organizadas em subpastas contidas de `"gen"`, na pasta do Google Drive colabs.

2.12 Revisão deste capítulo de Organização de Código

- Programas sequenciais
  - organize o seu código em três parte: > Entrada ⇒ Processamento ⇒ Saída
- Comentários
  - São úteis para outros podem ententer o seu código
- Desvios de fluxo
- Programas e subprogramas
- Funções, métodos e modularização
- Reaproveitamento e manutenção de código
  - Esses 4 últimos tópicos são muito importantes para organizar o seu código em partes
  - Fique atento ao **escopo** de uma variável **local** ou **global**
- Exercícios

2.13 Processando a Informação: Cap. 2: Organização de Código - Prática 1

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

2.13.1 Exercícios

Organizar cada questão em partes:

- **ENTRADA DE DADOS ⇒ PROCESSAMENTO DA INFORMAÇÃO ⇒ SAÍDA**

Seguindo o pseudocódigo a seguir:

```
# MINHA(S) FUNÇÃO(ÕES)
função delta(recebe: real a, real b, real c) retorna real d {
     d = b2 - 4ac
     retorne d
}

principal {
  # ENTRADAS
  a = 5
  b = -2
  c = 4

  # PROCESSAMENTO
  real valor = delta(a, b, c) # AQUI ESTÁ A CHAMADA DA FUNÇÃO

  # SAÍDA
  escreva("O delta de ax2 +bx + c é " + valor)
```

```
}
```

---

1. Crie um método que receba um valor inteiro qualquer e retorne 0 se este valor for par ou 1 se for ímpar (Dica: utilizar o operador resto %). Teste em um programa principal várias chamadas deste método.

---

2. Descreva o procedimento ou função para receber um ponto em coordenadas cartesianas (X, Y) e retornar a distância euclidiana até a origem (0, 0). Teste em um programa principal várias chamadas deste método.

---

3. Crie um método para calcular o ângulo formado entre um par de pontos X, Y, e o eixo x no plano cartesiano. Teste em um programa principal várias chamadas deste método.

---

4. Crie um método para calcular o ângulo formado entre um par de pontos X1, Y1 e X2, Y2 e o eixo x no plano cartesiano. Teste em um programa principal várias chamadas deste método.

---

5. Crie um programa com variáveis globais de um retângulo para base, altura e área. Crie no mesmo programa funções para calcular cada um dos 3 valores a partir dos outros 2: `calcula_base()`, `calcula_altura()` e `calcula_area()`. Teste em um programa principal várias chamadas deste método.

2.14 Processando a Informação: Cap. 2: Organização de Código - Prática 2

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve

ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

2.14.1 Exercícios

[Fonte: https://wiki.python.org.br/ExerciciosFuncoes]

Organizar cada questão em partes:

- **ENTRADA DE DADOS ⇒ PROCESSAMENTO DA INFORMAÇÃO ⇒ SAÍDA**

Seguindo o pseudocódigo a seguir:

```
# MINHA(S) FUNÇÃO(ÕES)
função delta(recebe: real a, real b, real c) retorna real d {
     d = b2 - 4ac
     retorne d
}

principal {
  # ENTRADAS
  a = 5
  b = -2
  c = 4

  # PROCESSAMENTO
  real valor = delta(a, b, c) # AQUI ESTÁ A CHAMADA DA FUNÇÃO

  # SAÍDA
  escreva(‘‘O delta de ax2 +bx + c é ’’ + valor)
}
```

---

1. Fazer uma função que recebe três argumentos e retorne o produto desses três argumentos. Teste em um programa principal várias chamadas deste método.

---

2. Faça um programa com uma função chamada `somaImposto`. A função possui dois parâmetros formais: taxaImposto, que é a quantia de imposto sobre vendas expressa em porcentagem e custo, que é o custo de um item antes do imposto. A função “altera” o valor de custo para incluir o imposto sobre vendas.

---

3. Faça um programa que use a função valorPagamento para determinar o valor a ser pago por uma prestação de uma conta. O programa deverá solicitar ao usuário o valor da prestação e o número de dias em atraso e passar estes valores para a função valorPagamento, que calculará o valor a ser pago e devolverá este valor ao programa que a chamou. O programa deverá então exibir o valor a ser pago na tela.

O cálculo do valor a ser pago é feito da seguinte forma. Considere que sempre tem atraso. Nestes casos, cobrar 3% de multa, mais 0,1% de juros por dia de atraso.

---

4. Escreva uma função que recebe dois inteiros, n e p, como parâmetros e retorna a **combinação** $\frac{n!}{p!(n-p)!}$. Use a função `math.factorial(x)` para calcular o fatorial de `x`. Conceitos:

- **Permutação** são agrupamentos de elementos de um conjunto nos quais a ordem dos elementos faz diferença. > Exemplo $\{a, b, c\} = abc, acb, bac, bca, cab, cba$. O número de combinações de um conjunto com n elementos é $n!$ (fatorial de n), onde $0! = 1$.
- **Combinação** indica quantas variedades de subconjuntos diferentes com $p \leq n$ elementos existem, onde a ordem dos elementos não interfere.

---

5. Cria uma função para ler três notas para prova1, prova2, projeto, declaradas como variáveis globais. Crie outra função para retornar a média ponderada com pesos, prova1 30%, prova2 40% e trabalho 30%. Crie uma terceira função para receber como parâmetros o nome de um aluno e a média e imprimir nesse formato:

```
Aluno: Ana Maria Chavier
Prova1: 7.0
Prova2: 8.0
Trabalho: 10.0
Média: X.0
```

3 Processando a Informação: Cap. 3: Desvios Condicionais

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

3.1 Sumário

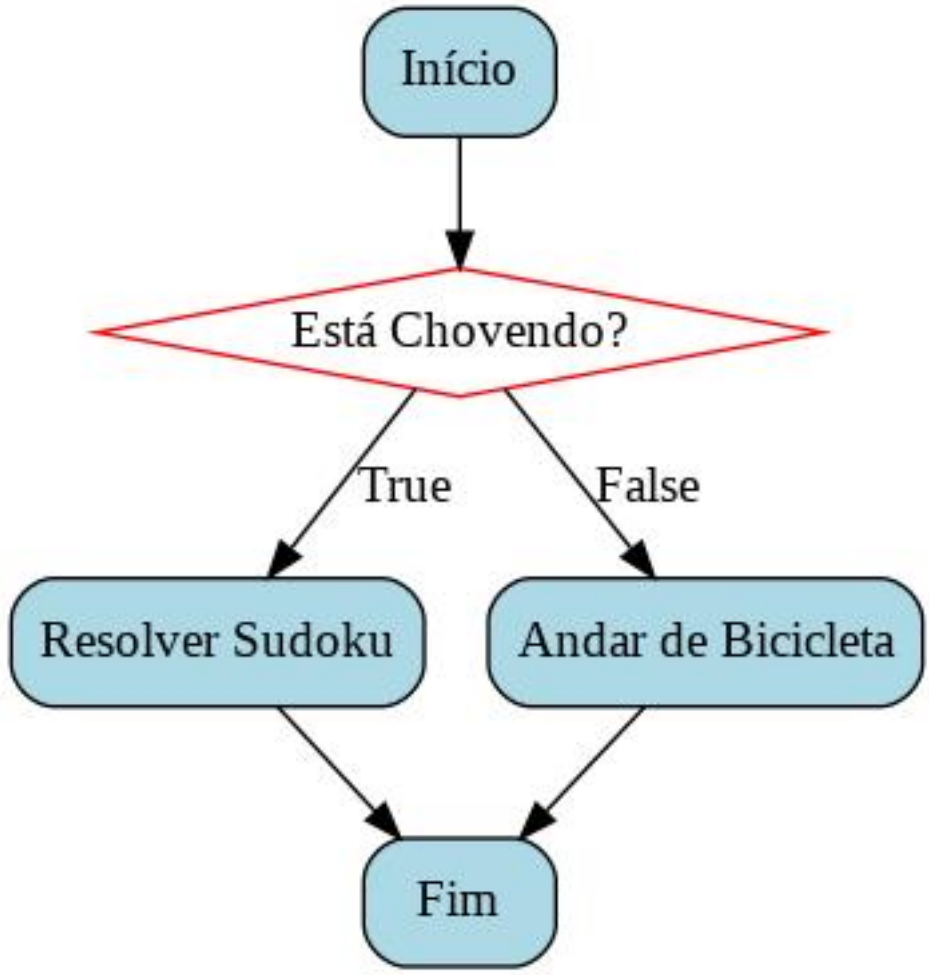
- Revisão do capítulo anterios
- O que é um desvio condicional?
- Condições com lógica booleana
- Desvios condicionais Simples e Compostos, com Encadeamentos
- Revisão deste capítulo
- Exercícios

3.2 Revisão do capítulo anterior (Organização de Código)

- No capítulo anterior foram apresentados formas de organização de códigos, utilizando comentários, tabulações, escopo de variáveis locais e globais, métodos (funções ou procedimentos) e o conceito de sistema de informação em partes:
  - **ENTRADA DE DADOS ⇒ PROCESSAMENTO DA INFORMAÇÃO ⇒ SAÍDA**
- Neste capítulo serão apresentados códigos com desvios condicionais, da forma:
  - `se` algo for verdade, `então`
    - faça algo1,
  - `senão` # essa parte é opcional
    - faça algo2.

3.3 O que é um Desvio Condional?

- O desvio condicional é a mais simples entre as estruturas lógicas não sequenciais em lógica de programação e fundamental para o entendimento de fluxo de código.
- A analogia básica com o processo de tomada de decisões ocorre quando imaginamos um cenário que proporciona duas possíveis alternativas de curso:
  - Se [condição] então faça [caminho caso verdadeiro] senão [caminho caso falso]
- Exemplos:
  - Se [está chovendo] então [resolver palavras cruzadas] senão [andar de bicicleta]
  - Se [é quarta] então [comer feijoada]
- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo):



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

```
[ ]: from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    if (Está Chovendo?) {
        Resolver Sudoku;
    } else {
        Andar de Bicicleta;
    }
    Fim;
    '''
)
```

3.4 Condições com Lógica *Booleana*

- O resultado de um teste condicional sempre resultará em um valor *booleano*, isto é:
  - com dois resultados possíveis: **verdadeiro** ou **falso**
  - Por convensão: $True = 1$ e $False = 0$
- Portanto, para condições, sempre usaremos combinações de operadores lógicos e relacionais para verificar o estado das variáveis verificadas. O seguinte pseudocódigo exemplifica algumas condições:

```
se vai chover, então leve um guarda-chuva.

se é feriado, então fique em casa.

se estou atrasado e está chovendo, então chame um taxi.

se minha nota é menor que 5, então fiquei de recuperação.
```

- Note que para todas as condições acima, a resposta para a condição é sempre: verdadeiro ou falso.
- Caso a condição seja verdadeira, será executada a operação ou operações especificadas na sequência.
- Codificando-se, as condições tomam a forma `se (condição) { comandos }`, como exemplos:

```
se (vai_chover) { leve um guarda-chuva }

se (feriado) { fique em casa }

se (atrasado e chovendo) { chame um taxi }

se (nota<5) { escrever "ficou de Recuperação" }
```

Esse código anterior **fica melhor se usar a organização** apresentada no capítulo anterior:

```
se (vai_chover)
  leve um guarda-chuva

se (feriado)
  fique em casa

se (atrasado e chovendo)
  chame um taxi

se (nota<5)
  escrever ‘‘ficou de Recuperação’’
```

- Observe que foram substituídas as chaves `{` e `}`, que definem os **escopos** das condicionais, pela tabulação. Isso é possível quando o bloco tem apenas uma instrução.
- Relembrando o Cap. 1 - Fundamentos, podemos usar nas **condicionais**, variáveis *booleanas* e operadores:
  - **relacionais**: ∘ == (é igual a) ∘ != (é diferente de) ∘ > (é maior que) ∘ < (é menor que) ∘ >= (é maior ou igual a) ∘ <= (é menor ou igual a)
  - Além de **lógicos**: ∘ && (e), & (e bit-a-bit) ∘ || (ou), | (ou bit a bit) ∘ ! (não lógico ou complemento) ∘ ~ (complemento bit-a-bit) ∘ ^ (ou exclusivo ‘XOR’ bit-a-bit) ∘ « N (shift-left, adiciona N zeros à direita do número em binário) ∘ » N (shift-right, elimina N dígitos a direita do número em binário)
- Os operadores **lógicos**, como os **relacionais**, sempre resultarão **verdadeiro** (V) ou **falso** (F), porém os operandos também são *booleanos*.
- Para as condições contendo AND, OR e XOR (ou exclusivo), as **tabelas verdade** para os dois operandos à esquerda e à direita, com valores lógicos representados na primeira linha e primeira coluna, são:

&&	V	F
V	**V**	F
F	F	F

||	V	F
V	V	V
F	V	**F**

^	V	F
V	F	V
F	V	F

- Logo,
  - os operandos devem ser ambos verdadeiros para que a operação AND retorne verdadeiro,

  - ao menos um deles verdadeiro para que o OR retorne verdadeiro e
  - ambos diferentes para que o XOR (ou exclusivo ^) retorne verdadeiro.

- Estas expressões, quando combinadas, resultarão sempre em um valor *booleano* V ou F, que pode ser então introduzido em um desvio condicional visando à realização de um subprograma.
- Por exemplo, usando os valores X=1, Y=2 e Z=4, qual o resultado das expressões abaixo?

  1. (X>0 && Y<2)
  2. (Z>0 || Z<5 && Y==4)
  3. (X»1 ==0 || Y«2>100)
  4. (X!=0 && Y!=0 && Z<0)
  5. (X=1)

- Dada a precedência de operadores estudada anteriormente e a combinação apresentada acima, apenas as expressões 2 e 3 resultariam em verdadeiro, dado $Z > 0$, assim como $1 >> 1$ (*shift* à direita uma casa) em binário é 0; são ambas condições suficientes para que o resultado seja verdadeiro.

[ ]:
```
X, Y, Z =1, 2, 4
equacao1 = (X>0 and Y<2)
equacao2 = (Z>0 or Z<5 and Y==4)
equacao3 = (X>>1 ==0 or Y<<2>100)
equacao4 = (X!=0 and Y!=0 and Z<0)
#equacao5 = (X=1) # ERRO de sintaxe, deveria ser X==1
print(equacao1,equacao2,equacao3,equacao4)
```

[ ]:
```
# exemplo de uso de ">>"
# 6 = 1 1 0 em binário => 1*2^2 + 1*2^1 + 0*2^0
6>>1 # = 3 = 0 1 1 em binário
```

- É importante ressaltar a diferença entre os operadores de comparação `==`, com leitura `é igual à`, e de atribuição `=`, tendo a leitura `recebe o valor de`. Neste aspecto, a expressão 5 está incorreta, já que o operador de atribuição não faz sentido quando usado desta forma.

3.5 Desvios Condicionais Simples e Compostos

```
se (condição) então faça
    Comandos

Volta para a parte sequencial
```

```
se (condição) então faça
    Comandos
senão faça
    Comandos

Volta para a parte sequencial
```

```
if (condição) {
    Comandos
}
```

```
if (condição) {
    Comandos
} else {
    Comandos
}
```

Exemplo 01 - Uso de Condicionais Simples

- Digamos que, como exemplo, desejamos calcular as raízes da equação de segundo grau usando a função `delta()` introduzida no capítulo anterior.
- Sabemos que as raízes dependem do sinal do Δ.
- Logo, a solução de uma equação do segundo grau se dá resolvendo as seguintes condições:
  1. Se $\Delta < 0$, x não possui raízes reais;
  2. Se $\Delta = 0$, x possui duas raízes reais idênticas;
  3. Se $\Delta > 0$, x possui duas raízes reais e distintas;
  4. Calcule as raízes, se existirem, usando as equações:

$$x1 = -b + \frac{\sqrt{\Delta}}{2a}$$

$$x2 = -b - \frac{\sqrt{\Delta}}{2a}$$

- Veja uma solução em pseudocódigo:

```
# a função é definida a seguir
função delta(a, b, c )
  retorne b * b - 4 * a * c

# programa principal
# ENTRADA DE DADOS
escreva("Calcula as raízes de equação de 2º grau: ax2 + bx + c")
real a = leia("Entre com o primeiro termo 'a': ")
real b = leia("Entre com o segundo  termo 'b': ")
real c = leia("Entre com o terceiro termo 'c': ")

# PROCESSAMENTO E SAÍDA
real d = delta(5, -2, 4)
escreva(''O delta é '' + valor)
se (d < 0)
  escreva("A equação não possui raízes reais")
se (d == 0)
  escreva("A raíz é " + (-b + raíz(d)/2*a))
```

```
se (d > 0)
  escreva("As raízes são x1=" + (-b - raíz(d)/2*a)) + " e x2="  +
  (-b + raíz(d)/2*a))
```

Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=3
1
4
output=
O delta é  -47.0
A equação não possui raízes reais.
case=caso2
input=4
6
2
output= O delta é 4.0
Raízes: -10.0 e -2.0.
```

[ ]:
```
%%writefile cap3ex01.c
#include <stdio.h>
#include <math.h>
float delta(float a, float b, float c) {
  float d = b*b-4*a*c;
  return d;
}

float leia() {
  float valor;
  printf("Entre com um valor: ");
  scanf("%f", &valor);
  return valor;
}

int main(void) {

  // ENTRADAS
  float a, b, c;
  a = leia();
  b = leia();
  c = leia();

  // PROCESSAMENTO e SAÍDA
  double d = (double) delta(a, b, c);
  printf("Delta = %.1f\n", d);
  if (d < 0) {
```

```
        printf("A equação não possui raízes reais");
    }
    if (d == 0) {
        printf("Raíz: %.1f",(-b + sqrt(d) / 2 * a));
    }
    if (d > 0) {
        printf("Raíz: %.1f e %.1f ",(-b - sqrt(d) / 2 * a), (-b + sqrt(d)␣
 ↪/ 2 * a));
    }
    return 0;
}
```

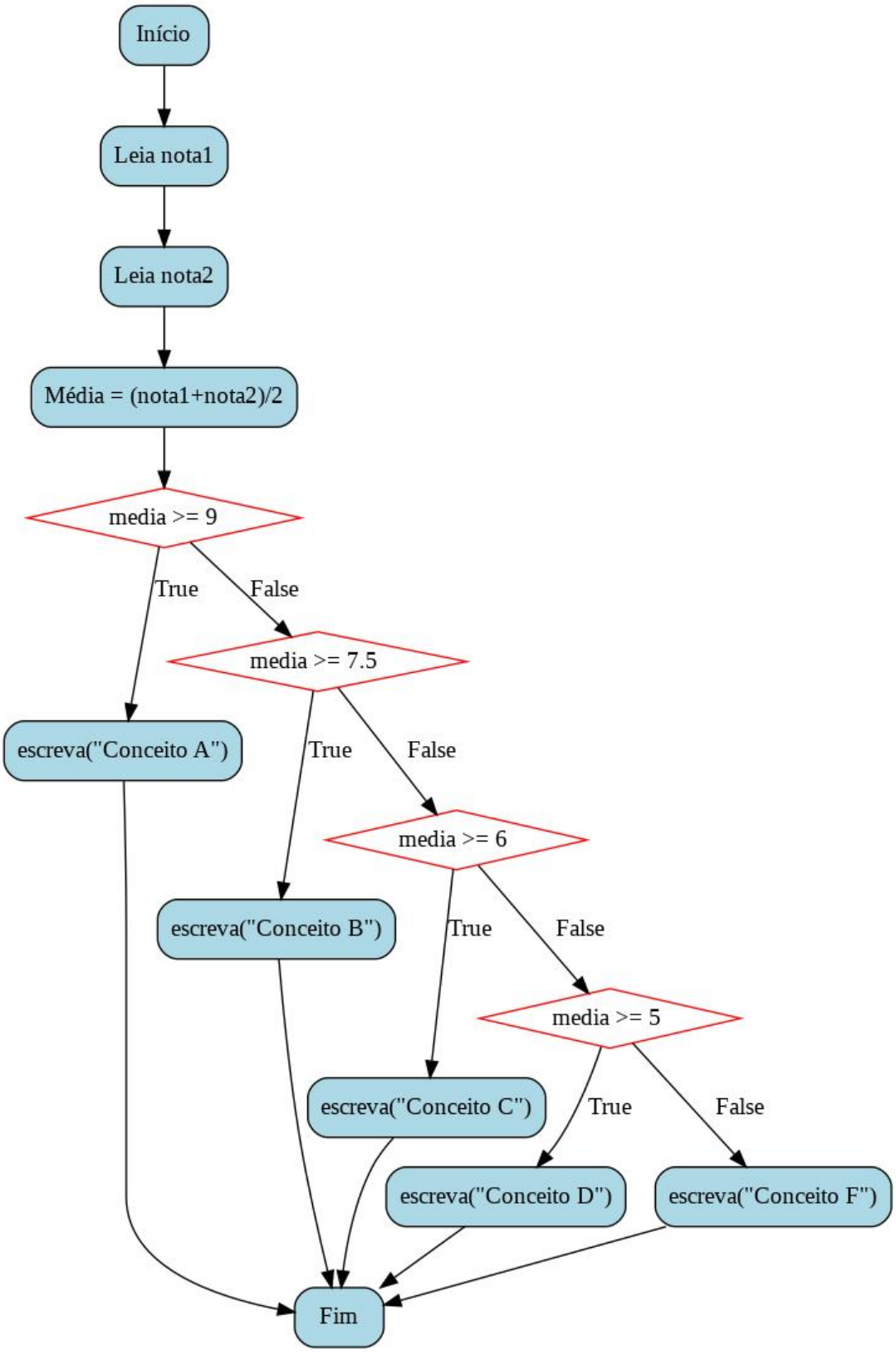
[ ]:
```
%%shell
gcc -Wall -std=c99 cap3ex01.c -o output2 -lm
./output2
# A biblioteca matemática deve ser vinculada ao construir o executável.
# Como fazer isso varia de acordo com o ambiente,
# mas no Linux / Unix, basta adicionar -lm ao comando
```

Exemplo 02 - Uso de Condicionais Compostas e Encadeadas

- Considere o seguinte pseudocódigo para ler duas notas reais, calcular a média das duas notas e atribui um conceito:

```
nota1 = leia("Digite a 1a nota:");
nota2 = leia("Digite a 2a nota:");
media = (nota1 + nota2)/2;
se media >= 9 então
    escreva("Conceito A");
senão se media >= 7.5
    escreva("Conceito B");
senão se media >= 6
    escreva("Conceito C");
senão se media >= 5|
    escreva("Conceito D");
senão
    escreva("Reprovado! Conceito F");
```

- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo):



[ ]:
```
from txtoflow import txtoflow
```

```
txtoflow.generate(
    '''
    Início;
    Leia nota1;
    Leia nota2;
    Média = (nota1+nota2)/2;
    if ( media >= 9 ) {
        escreva("Conceito A");
    } else if (media >= 7.5 ) {
        escreva("Conceito B");
    } else if (media >= 6 ) {
        escreva("Conceito C");
    } else if (media >= 5 ) {
        escreva("Conceito D");
    } else {
        escreva("Conceito F");
    }
    Fim;
    '''
)
```

Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 02, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=4.0
6.0
output=
Conceito D
case=caso2
input=4.0
5.0
output=
Conceito F
case=caso2
input=5.0
7.0
output=
Conceito C
case=caso3
input=6.0
9.0
output=
Conceito B
case=caso4
input=9.0
10.0
```

```
output=
Conceito A
```

[ ]:
```
%%writefile cap3ex02.c
#include <stdio.h>
float leia() {
  float valor;
  printf("Entre com um valor: ");
  scanf("%f", &valor);
  return valor;
}

int main(void) {

  // ENTRADAS
  float nota1 = leia();
  float nota2 = leia();

  // PROCESSAMENTO E SAÍDA
  float media = (nota1 + nota2)/2;
  if (media >= 9.0)
    printf("Conceito A");
  else if (media >= 7.5)
    printf("Conceito B");
  else if (media >= 6.0)
    printf("Conceito C");
  else if (media >= 5.0)
    printf("Conceito D");
  else
    printf("Reprovado! Conceito F.");
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap3ex02.c -o output2
./output2
```

3.6 Comando `switch`

O `switch` verifica se uma variável (do tipo `int` ou `char`) é ou não igual a certo valor constante (valor1, valor2, ..., valorN na sintaxe a seguir):

```
switch (variável) {
  case valor1:
    Comandos;
    break;
  case valor2:
    Comandos;
    break;
```

```
  case valorN:
    Comandos;
    break;
  default: // opcional, caso não ocorram os casos anteriores
    Comandos;
}
```

Esse comanto `switch` é útil quando se tem um menu de opções a ser executado.

Exemplo 03 - Exemplo de `switch` com menu de opções

[1]:
```
%%writefile cap3ex03.c
#include <stdio.h>
int main() {
  float area;
  int num;
  char escolha;
  printf("1. Circulo\n");
  printf("2. Quadrado\n");
  printf("Escolha:\n");

  scanf("%c", &escolha);

  switch (escolha) {
  case '1':
    printf("Raio:\n");
    scanf("%d", &num);
    area = 3.14 * num * num;
    printf("Area do circulo: ");
    printf("%.2f\n", area);
    break;

  case '2':
    printf("Lado:\n");
    scanf("%d", &num);
    area = num * num;
    printf("Area do quadrado: ");
    printf("%.2f\n", area);
    break;

  default:
    printf("Escolha Incorreta!\n");
  }
  return 0;
}
```

[3]:
```
%%shell
gcc -Wall -std=c99 cap3ex03.c -o output3
./output3
```

3.7 Exercícios

Ver notebook Colab no arquivo `cap3.part2.lab.*.ipynb` (`*` é a extensão da linguagem), utilizando aluma linguagem de programação de sua preferência, onganizadas em subpastas contidas de `"gen"`, na pasta do Google Drive colabs.

3.8 Revisão deste capítulo de Desvios Condicionais

- O que é um desvio condicional?
- Condições com lógica booleana
- Desvios condicionais Simples e Compostos, com Encadeamentos
- Exercícios
- Revisão deste capítulo de Desvios Condicionais

3.9 Processando a Informação: Cap. 3: Desvios Condicionais - Prática 1

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

3.9.1 Exercícios

---

1. Crie um método com apenas condicionais e o operador resto (%) para:

- Determinar se um número entrado pelo teclado é par ou ímpar, exibindo a mensagem apropriada na tela;

- Modifique o método anterior para verificar se o número entrado é múltiplo de 3.

Teste em um programa principal várias chamadas destes métodos.

---

2. Faça um programa que leia (peça para o usuário digitar) três números inteiros quaisquer, armazenando nas variáveis A, B e C e imprima os números em ordem do menor para o maior.

---

3. Faça um programa que receba três valores inteiros nas variáveis A, B e C e ordene os valores nas próprias variáveis, de forma que, no final da execução, a variável A contenha o menor valor e C o maior valor. O programa deve usar apenas 4 variáveis: A, B, C e T.

---

4. Faça um programa em qualquer linguagem para determinar a classificação do peso de um indivíduo, de acordo com a tabela:

Tabela: IMC = peso / altura2	
Magro	IMC até 18,5
Saudável	IMC até 25,0
Acima do peso	IMC até 30,0
Obeso	IMC até 35,0
Morbidez	IMC 35 mais

---

5. Faça um programa para ler três notas (nota1, nota2 e nota3, com pesos 3, 3 e 4, respectivamente), calcular a média ponderada, fazer a conversão para conceito, conforme critérios definidos no Cap. 3.

3.10 Processando a Informação: Cap. 3: Desvios Condicionais - Prática 2

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve

ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

3.10.1 Exercícios

[Fonte: link]

---

1. Fazer um programa que leia a capacidade de um elevador e o peso de 5 pessoas. Informar se o elevador está liberado para subir ou se excedeu a carga máxima.

---

2. Faça um programa que leia 4 variáveis A, B, C e D. A seguir, se B for maior do que C e se D for maior do que A e a soma de C com D for maior que a soma de A e B e se C e D, ambos, forem positivos e se a variável A for par, escrever a mensagem "valores aceitos", senão escrever "valores não aceitos"

---

3. Escreva um programa que informe se um dado ano é ou não é bissexto. Um ano é bissexto se ele for divisível por 400 ou se ele for divisível por 4 e não por 100.

---

4. Construa um programa que leia um número inteiro e imprima a quantidade de centenas, dezenas e unidades desse número. Considere o seguinte exemplo: o número 345 possui 3 centenas, 4 dezenas e 5 unidades.

---

5. Faça um programa que pergunte o nome do aluno, a quantidade de dias na semana e o tipo de curso (B para básico, I para intermediário e A para avançado). Mostre o nome do aluno e o valor a ser pago. O valor total é calculado com base nas informações abaixo:

- Caso a opção escolhida for Básico, deverá fazer a seguinte conta: > Valor Total = (Quantidade de dias na semana * 7) * 15
- Caso a opção escolhida for Intermediário, deverá fazer a seguinte conta: > Valor Total = (Quantidade de dias na semana * 8,5) * 20
- Caso a opção escolhida for Avançado, deverá fazer a seguinte conta: > Valor Total = (Quantidade de dias na semana * 10) * 25

3.11 Processando a Informação: Cap. 3: Desvios Condicionais - Prática 3

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

3.11.1 Exercícios

[Fonte: link]

---

1. Faça um programa que leia 2 notas de um aluno, verifique se as notas sao válidas e exiba na tela a média destas notas. Uma nota válida deve ser, obrigatoriamente, um valor entre 0.0 e 10.0, onde caso a nota não possua um valor válido, este fato deve ser informado ao usuário e o programa termina.

---

2. Escreva um programa que leia um número inteiro maior do que zero e devolva, na tela, a soma de todos os seus algarismos. Por exemplo, ao numero 251 corresponder ao valor 8 (2 + 5 + 1). Se o numero lido não for maior do que zero, o programa terminará com a mensagem "Numero inválido".

---

3. Escreva um programa que leia um inteiro entre 1 e 12 e imprima o mês correspondente a este número. Isto é, janeiro se 1, fevereiro se 2, e assim por diante.

---

4. Faça uma função que calcule e retorne a área de um trapézio (A). Lembre-se que a base maior e a base menor devem ser números maiores que zero.

$$A = \frac{altura * (basemaior + basemenor)}{2}$$

---

5. Faça um programa para verificar se um determinado número inteiro e divisível por 3 ou 5, mas nao simultaneamente pelos dois.

---

6. Escreva o menu de opções abaixo. Leia a opção do usuário e execute a operação escolhida. Escreva uma mensagem de erro se a opção for inválida.

```
Escolha a opção:
1- Soma de 2 números.
2- Diferença entre 2 números (maior pelo menor).
3- Produto entre 2 números.
4- Divisão entre 2 números (o denominador não pode ser zero).
Opção
```

---

7. Leia a idade e o tempo de serviço de um trabalhador e escreva se ele pode ou não se aposentar. As condições para aposentadoria são:

- Ter pelo menos 65 anos,
- Ou ter trabalhado pelo menos 30 anos,
- Ou ter pelo menos 60 anos e trabalhado pelo menos 25 anos.

---

8. Uma empresa vende o mesmo produto para quatro diferentes estados. Cada estado possui uma taxa diferente de imposto sobre o produto (MG 7%; SP 12%; RJ 15%; MS 8%). Faça um programa em que o usuario entre com o valor e o estado destino do produto e o programa retorne o preço final do produto acrescido do imposto do estado em que ele será vendido. Se o estado digitado não for válido, mostrar uma mensagem de erro.

---

9. Leia a distância em Km e a quantidade de litros de gasolina consumidos por um carro em um percurso, calcule o consumo em Km/l e escreva uma mensagem de acordo com a tabela abaixo:

Consumo	Km/l	Mensagem
menor que	8	venda o carro!
entre	8 e 14	econômico!
maior que	14	super econômico!

---

10. Escreva um programa que, dada a idade de um nadador, classifique-o em uma das seguintes categorias:

Categora	Idade
Infantil A	5 a 7
Infantil B	8 a 10
Juvenil A	11 a 13
Juvenil B	14 a 17
Sênior	maior que 17

---

11. Escrever um programa que leia o codigo do produto escolhido do cardápio de uma lanchonete e a quantidade. O programa deve calcular o valor a será pago por aquele lanche. Considere que a cada execução somente será calculado um pedido. O cardápio da lanchonete segue o padrao abaixo:

Especificação	Código	Preço
Cachorro quente	100	12.0
Bauru Simples	101	13.0
Bauru Simples	102	13.0
Hamburguer	103	12.0
Cheeseburguer	104	17.0
Suco	105	6.0
Refrigerante	106	4.0

4 Processando a Informação: Cap. 4: Estruturas de Repetição (Laços)

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

4.1 Sumário

- Revisão do capítulo anterios
- Quando usar repetições?
- Tipos de estruturas de repetição
- Laços aninhados
- Validação de dados com laços
- Interrupção da execução dos laços
- Recursão
- Revisão deste capítulo
- Exercícios

4.2 Revisão do capítulo anterior (Desvios Condicionais)

- No capítulo anterior foram apresentadas formas de construir código contendo estruturas condicionais simples ou compostas. Ou seja, desvios condicionais, da forma:
  - `se` algo for verdade, `então`
    - faça algo1,
  - `senão` # essa parte é opcional
    - faça algo2.
- Neste capítulo iremos abordar as **Estruturas de Repetição**, conhecidas também como **Laços** ou ***Loops***.

4.3 Quando usar repetições?

- As estruturas de repetição são recomendadas para quando um padrão de código é repetido várias vezes sequencialmente, apenas alterando-se o valor de uma ou mais variáveis entre os comandos repetidos.
- Veja no exemplo a seguir um pseudocodigo, para imprimir a tabuada de um número `t` entrado pelo usuário no formato:

Tabuada x (número de 1 a 10) = valor

```
Inteiro: t, n = 1;
t = leia("Qual a tabuada? ");
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
```

- Embora o código não pareça extenso, é fácil imaginar uma situação onde tenhamos que repetir 100, 500, 1000 vezes um mesmo bloco de instruções.

4.4 Tipos de estruturas de repetição

4.4.1 Pseudocódigo

faça-enquanto

```
faça {
  comandos
} enquanto (condição);
```

enquanto-faça

```
enquanto (condição) faça {
  comandos
}
```

para

```
para variável = valor_inicial até valor_final, variável++, faça {
  comandos
}
```

para reverso

```
para variável = valor_final até valor_inicial, variável--, faça {
```

```
  comandos
}
```

- Note que no caso do `enquanto-faça` é necessário que a condição seja verdadeira para que os comandos presentes no bloco de execução sejam processados.
- Neste caso, se ao entrar no comando enquanto (*while*) a condição do teste for falsa, oposto ao `faça-enquanto`, o subprograma não será executado.
- Isto é, todo o código dentro do bloco do laço será pulado já na verificação inicial da condição no `enquanto-faça`, seguindo diretamente para a parte sequencial subsequente, similar ao que ocorre no se-então.
- A forma `faça-enquanto` é recomendada quando queremos que os comandos contidos no laço sejam executados ao menos uma vez, mesmo que a condição seja inicialmente falsa.
- O laço `para` é recomendado quando se sabe o número de iterações existes (quantas vezes o bloco dentro do laço será executado). Por exempo, no caso anterior do problema da Tabuada.

4.4.2 C/CPP/Java/JavaScript

do-while

```
do {
  comandos;
} while (condição);
```

while

```
while (condição) {
  comandos;
}
```

for

```
for(v=0; v<10; v++) {
  comandos;
}
```

Em algumas linguagens de programação é possível omitir um ou todos os parâmetros, por exemplo: `for(;;) {...}`.

4.4.3 Pseudocódigo: Exemplo laço faça-enquanto.

```
Real: nota, média, acumulador=0, contador=0;
Caractere: resposta='lixo';

faça {
   nota = leia("Entre com uma nota: ");
   acumulador = acumulador + nota;
   contador = contador + 1;
```

```
   resposta = leia("Deseja continuar? (s/n): ");
} enquanto (resposta == 's');

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- Além do `contador`, o programa usa um `acumulador` (variável que acumula as notas digitadas).
- Repare que a condição `resposta == 's'` no `faça-enquanto` é falsa até que seja efetuada a leitura da variável resposta dentro do laço, em: `resposta = leia("Deseja continuar? (s/n): ");`
- Apenas do caso de o usuário entrar com o caractere `'s'`, o laço será repetido novamente.
- Isto quer dizer que o estado da condição é falso na primeira execução do código do laço.

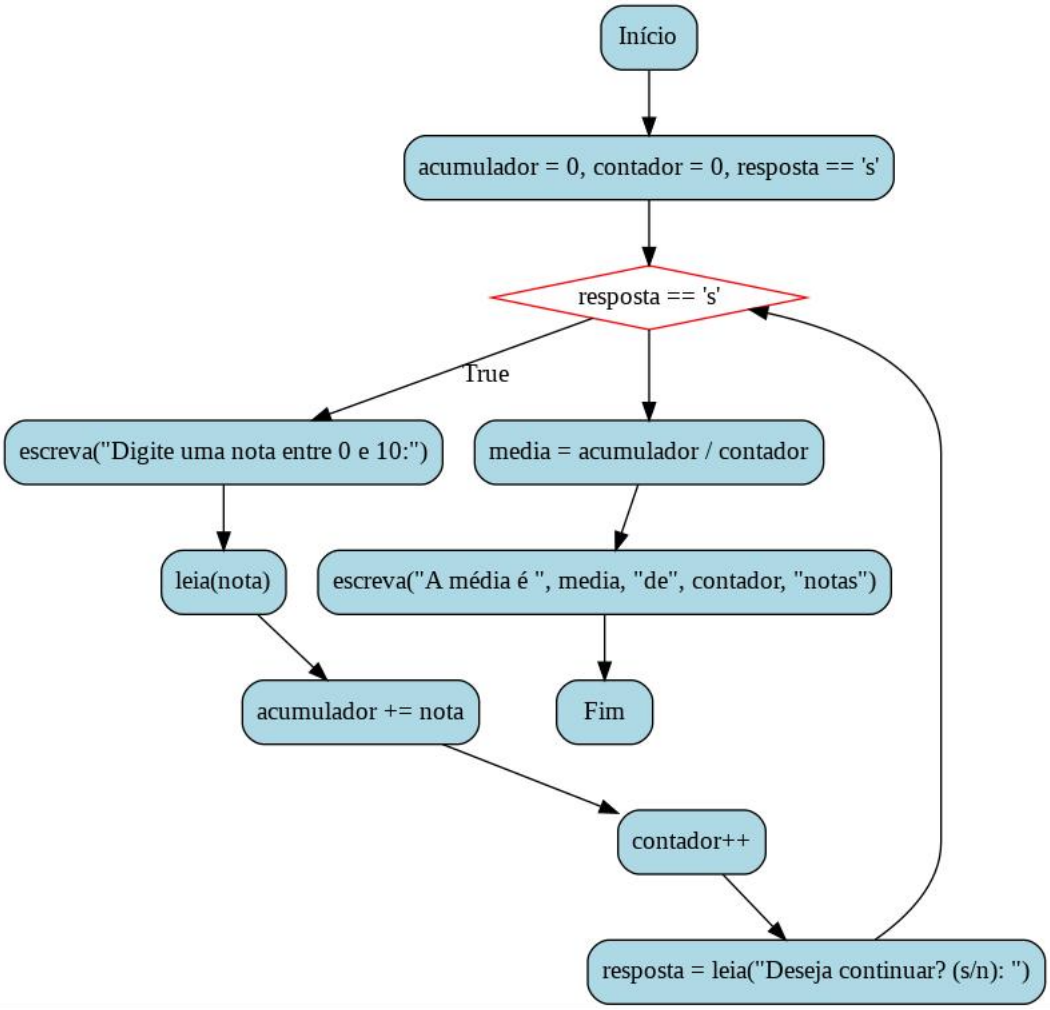
4.4.4 Pseudocódigo: Exemplo laço enquanto-faça.

```
Real: nota, média, acumulador=0, contador=0;
Caractere: resposta='s';

enquanto (resposta == 's') faça {
   nota = leia("Entre com uma nota: ");
   acumulador = acumulador + nota;
   contador = contador + 1;
   resposta = leia("Deseja continuar? (s/n): ");
}

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- Observe que no pseudocódigo anterior do `enquanto-faça` temos o mesmo resultado do `faça-enquanto`, pois a variável `resposta` é inicializada com `s`, satisfazendo a condição lógica e entrando no laço.
- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo):



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    acumulador = 0, contador = 0, resposta == 's';
    while (resposta == 's') {
      escreva("Digite uma nota entre 0 e 10:");
      leia(nota);
      acumulador += nota;
      contador++;
      resposta = leia("Deseja continuar? (s/n): ");
    }
    media = acumulador / contador;
    escreva("A média é ", media, "de", contador, "notas");
```

```
    Fim;
    '''
)
```

4.4.5 Pseudocódigo: Exemplo laço enquanto-faça INFINITO.

Considere uma alteração no código anterior para não fazer mais a pergunta `Deseja continuar? (s/n)`, mas entrando num laço `enquanto-faça` para ler e acumular 100 notas:

```
Real: nota, média, acumulador=0,contador=0;

enquanto (contador < 100) faça {
   nota = leia("Entre com uma nota: ");
   acumulador = acumulador + nota;
   # contador = contador + 1;
}

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- O que vai ocorrer ao escrever e rodar esse código em alguma linguagem de programação?
- Onde está o erro?
- **MUITO CUIDADO COM LAÇOS INFINITOS EM ATIVIDADES NO MOODLE+VPL!** Se a execução demorar mais que 1 minuto, provavelmente entrou em um laço infinito e terá que recarregar a página.
- Esta condição, onde a execução fica "presa" dentro do laço, é conhecida como ***DEADLOCK***.
- *Deadlocks* geram erro de finalização de programa, que executará eternamente, podendo travar o programa, o teclado e o mouse ou até mesmo o computador, neste caso, sendo necessário um ***RESET*** para sair do laço.

4.5 Validação de Dados

- Uma possível aplicação de laços é garantir que os dados entrados sejam válidos.
- **Validação de dados** é o nome dado à verificação dos valores de entrada, se os mesmos se encontram dentro dos limites previstos ou no formato adequado, notificando o usuário no caso de valores inválidos.
- O exemplo a seguir é o pseudocódigo apresentado anteriormente, incorporando a validação de dados de entrada (nota entre 0 e 10), indicando o erro e pedindo para o usuário entrar novamente o dado até que seja válido.

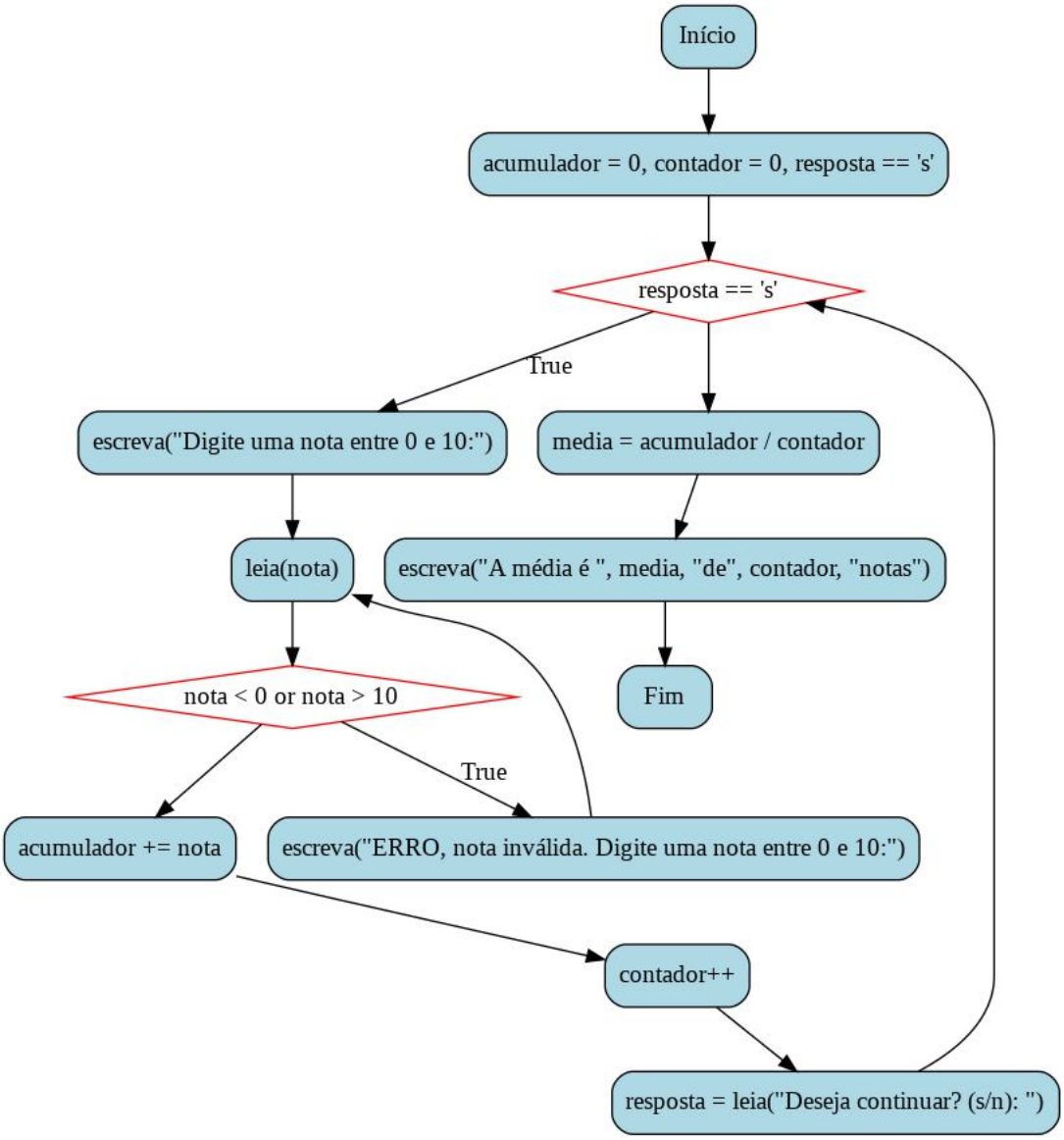
4.5.1 Pseudocódigo: Exemplo laço faça-enquanto, com validação

```
Real: nota, média, acumulador=0, contador=0;
Caractere: resposta='lixo';
```

```
faça {
  faça {
    nota = leia("Entre com uma nota entre 0 e 10: ");
    se (nota < 0 || nota > 10) então
      escreva("ERRO, nota inválida. Digital nota entre 0 e 10!")
  } enquanto (nota < 0 || nota > 10);
  acumulador = acumulador + nota;
  contador = contador + 1;
  resposta = leia("Deseja continuar? (s/n): ");
} enquanto (resposta == 's');

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- No pseudocódigo anterior, o segundo `faça-enquanto` aceita apenas notas entre 0 e 10. Caso contrário, escreve uma mensagem de erro e solicita nova nota.
- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo).
- Observar que essa biblioteca `txtoflow` não aceita `faça-enquanto`, assim como o Python.



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    acumulador = 0, contador = 0, resposta == 's';
    while (resposta == 's') {
      escreva("Digite uma nota entre 0 e 10:");
      leia(nota);
      while (nota < 0 or nota > 10) {
        escreva("ERRO, nota inválida. Digite uma nota entre 0 e 10:");
```

```
        leia(nota);
      }
      acumulador += nota;
      contador++;
      resposta = leia("Deseja continuar? (s/n): ");
    }
    media = acumulador / contador;
    escreva("A média é ", media, "de", contador, "notas");
    Fim;
    '''
)
```

4.6 Interrupção dos laços

- Algumas linguagens permitem interromper a execução do laço através do comando 'interromper' ou 'quebrar' (**break**).
- Isto pode ser útil caso se queira interromper o laço em algum evento específico.

4.7 Exemplo 01 - Ler 10 notas com validação

Considere um algoritmo para ler 10 notas válidas, entre 0 e 10, escrevendo no final a média nas notas válidas lidas.

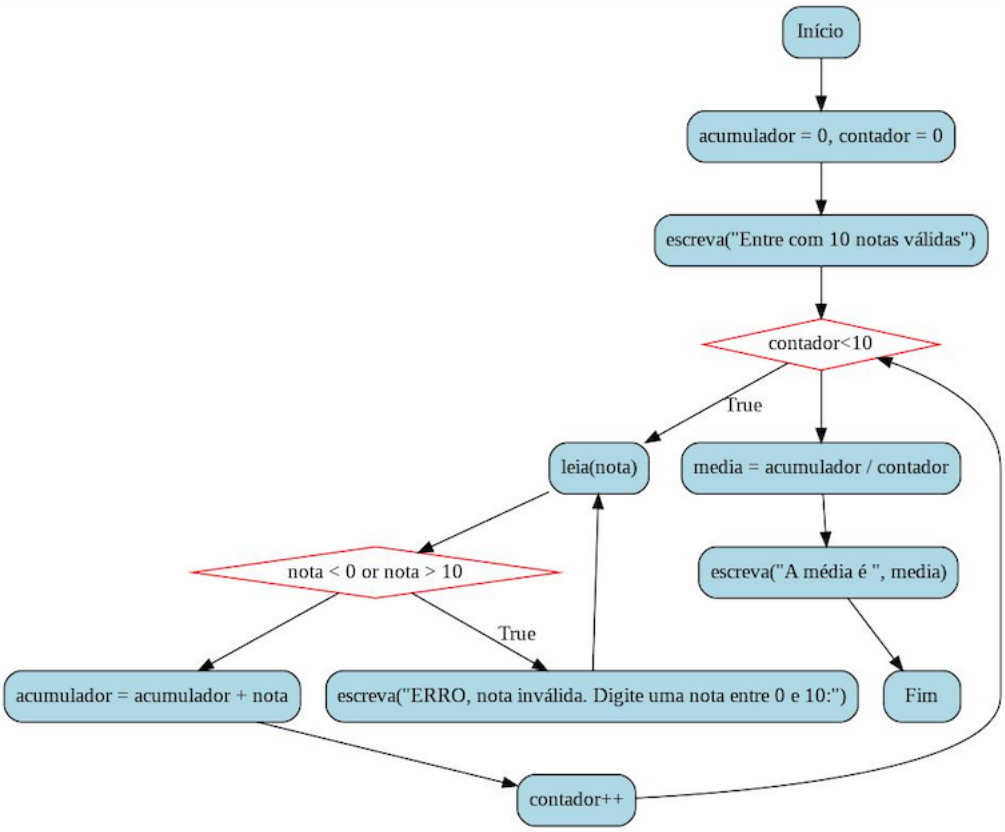
Pseudocódigo

```
Real: nota, média, acumulador=0,contador=0;

escreva("Entre com 10 notas válidas")
faça {
  faça {
    nota = leia();
  } enquanto (nota < 0 || nota > 10);
  acumulador = acumulador + nota;
  contador = contador + 1;
} enquanto (contador < 10);

média = acumulador / contador;
escreva("A média das " + contador + " notas é " + média);
```

- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo):



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    acumulador = 0, contador = 0;
    escreva("Entre com 10 notas válidas");
    while (contador<10) {
      leia(nota);
      while (nota < 0 or nota > 10) {
        escreva("ERRO, nota inválida. Digite uma nota entre 0 e 10:");
        leia(nota);
      }
      acumulador = acumulador + nota;
      contador++;
    }
    media = acumulador / contador;
    escreva("A média é ", media);
    Fim;
    '''
```

```
)
```

Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=9.0
78.0
6.0
9
8
7
6
5
7
8
6
output=
A média das 10 notas é 7.1
case=caso2
input=9.0
78.0
-9.0
9876.0
9876.0
7.0
9
8
4
6
6
8
9
7
output=
A média das 10 notas é 7.3
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[ ]:
```
%%writefile cap4ex01.c
#include <stdio.h>

int main(void) {

  // ENTRADA DE DADOS e PROCESSAMENTO
  float acumulador = 0, nota, media;
```

```
  int contador = 0;

  printf("Entre com 10 notas válidas\n");
  while (contador<10) {
    do {
        scanf("%f", &nota);
    } while (nota < 0.0 || nota > 10.0 );
    acumulador = acumulador + nota;
    contador++;
  }
  media = acumulador/contador;

  // SAÍDA
  printf("A média das %d notas é %.1f\n", contador, media);

  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap4ex01.c -o output
./output
```

4.8 Recursão

Este tópico de recursão (ou recursividade) é complementar ao livro, em sua primeira edição.

Como nos laços de repetição, a recursão tem como objetivo rodar trechos de códigos (agora encapsulados em métodos) várias vezes.

Além disso, análogo ao laço, **muito cuidado com o critério de parada, senão o código irá fazer infinitas chamadas recursivas, podendo travar o seu computador!**

Ou seja, o método recursivo teve ter pelo menos uma condicional e argumentos que variam nas chamadas recursivas que garantam a convergência (critério de parada).

As funções a seguir atendem a esses requisitos? Senão, como corrigir? O que elas fazem?

```
int funcao_recursiva(int a) {
    if (a == 0) return a;
    return 1+funcao_recursiva(a-1);
}
int funcao_recursiva(int a) {
    if (a == 0) return a;
    return 1+funcao_recursiva(a-2);
}
```

Veja mais um exemplo a seguir.

4.9 Exemplo 02 - Fatorial

Considere um algoritmo para calcular o fatorial de n [ref].

Pseudocódigo

```
# MINHA FUNÇÃO RECURSIVA
função fatorial(recebe: inteiro n) retorna inteiro {
  se (n == 1) faça { # CRITÉRIO DE PARADA
    retorne 1
  }
  retorne n * fatorial(n-1);
}

Inteiro: n;
escreva("Entre com numero: " + n)
escreva("Fatorial de " + n + " é " + fatorial(n) );
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[16]:
```
%%writefile cap4ex02.c
#include <stdio.h>

int fatorial(int n) {
  long int f;
  printf("debug - antes  - n = %d\n",n);
  if (n == 1) f = 1; // CRITÉRIO DE PARADA!!!
  else // CHAMADA RECURSIVA, COM ALTERAÇÃO DO VALOR DO ARGUMENTO!!!
    f = n * fatorial(n-1);
  printf("debug - depois - n = %d; fatorial = %ld\n", n, f);
  return f;
}

int main(void) {
  // ENTRADA DE DADOS e PROCESSAMENTO
  int n;

  printf("Entre com n: ");
  scanf("%d", &n);

  // PROCESSAMENTO E SAÍDA
  printf("Fatorial de %d é %d\n", n, fatorial(n));

  return 0;
}
```

[17]:
```
%%shell
gcc -Wall -std=c99 cap4ex02.c -o output
./output
```

4.10 Exemplo 03 - Ler 10 notas, com recursão

Considere um algoritmo para ler 10 notas válidas utilizando recursão e calcular a média.

Pseudocódigo

```
# MINHA FUNÇÃO RECURSIVA
função lerNota(recebe: real acumulador, inteiro n) retorna real acumulador {
  Real: nota;
  se (n > 0) faça { # CRITÉRIO DE PARADA
    faça {
      nota = leia();
    } enquanto (nota < 0 || nota > 10);
    acumulador = lerNota(acumulador, n - 1); # CHAMADA RECURSIVA,
    # COM ALTERAÇÃO DO VALOR DO ARGUMENTO !!!
  }
  retorne acumulador + nota;
}

Real: média, acumulador=0, contador=10;

escreva("Entre com " + contador + " notas válidas")
acumulador = lerNota(acumulador, contador); # CHAMDADA RECURSIVA!!!

média = acumulador / contador;
escreva("A média das " + contador + " notas é " + média);
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[ ]:
```c
%%writefile cap4ex03.c
#include <stdio.h>

float lerNota(float acumulador, int n) {
  float nota;
  if (n > 0) { // CRITÉRIO DE PARADA!!!
    do {
      scanf("%f", &nota);
    } while (nota < 0.0 || nota > 10.0);
    // CHAMADA RECURSIVA, COM ALTERAÇÃO DO VALOR DO ARGUMENTO!!!
    acumulador = lerNota(acumulador, n - 1);
  }
  return acumulador + nota;
}

int main(void) {

  // ENTRADA DE DADOS e PROCESSAMENTO
  float acumulador = 0, media;
  int contador = 10;
```

```
  printf("Entre com %d notas válidas\n", contador);

  // CHAMADA RECURSIVA
  acumulador = lerNota(acumulador, contador);

  media = acumulador / contador; // MÉDIA

  // SAÍDA
  printf("A média das %d notas é %.1f\n", contador, media);

  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap4ex03.c -o output
./output
```

4.11 Exercícios

Ver notebook Colab no arquivo `cap4.part2.lab.*.ipynb` (`*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, onganizadas em subpastas contidas de `"gen"`, na pasta do Google Drive colabs.

4.12 Revisão deste capítulo de Estruturas de Repetição (Laços)

- Quando usar repetições? > Quando existem instruções que se repentem.
- Tipos de estruturas de repetição:
  - Depende da linguagem. Algumas possibilidades:
    - `do-while` (não aceita em Python)
    - `while` (todas)
    - `for` (todas)
    - `repeat` (R)
  - Qual usar?
    - Depende da lógica ser implementada e da linguagem utilizada.
    - Se tiver um número fixo de iterações, geralmente se usa `for`.
- Laços aninhados
- Validação de dados com laços
  - Incluir um laço para verificar o valor lido.
- Interrupção da execução dos laços
  - Depende da linguagem, algumas possibilidades:
    - `break` - interrompe o laço
    - `continue` - não executa o final do laço
    - `exit` - aborta o laço e o programa!
- Outra forma de executar trechos de códigos várias vezes é encapsular em métodos recursivos.
  - Atenção com o critério de parada!
  - Atenção com os argumentos do método recursivo!

- Exercícios
- Revisão deste capítulo de Estruturas de Repetição (Laços)

4.13 Processando a Informação: Cap. 4: Estruturas de Repetição (Laços) - Prática 1

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

4.13.1 Exercícios

---

1. Demonstre o uso de laços imprimindo os 50 primeiros múltiplos de 3.

---

2. Faça um método `fibonacci(n)` ou `F(n)` para retornar o Fibonacci $F(n)$, definido por $F(0) = 0$, $F(1) = 1$, e $\forall n > 1$:

$$F(n) = 0, 1, 1, 2, 3, 5, \cdots, F(n-2), F(n-1), F(n-1) + F(n-2)$$

   Teste em um programa principal várias chamadas destes métodos.

---

3. Faça um método `primo(n)` para retornar o valor `1` (número um) se um número n entrado pelo teclado é primo, caso contrário, retornar o valor `0` (número zero). Isto pode ser feito dividindo sucessivamente o número entrado por valores i, onde i varia de 2 até $n-1$, e verificando o resto da divisão. Se $n\%i$ (resto da divisão de n por i) for zero para qualquer i, o método dever retornar o valor `0`. Caso a condição anterior não ocorra, o método dever retornar o valor `1`.

Teste em um programa principal várias chamadas destes métodos, exibindo a mensagem `"Não é primo!"` ou `"é primo"`.

---

4. Escreva um programa que

- leia três dados: Investimento inicial (I), Taxa de juros (J) e Número de meses (N);
- em seguida calcule e exiba uma tabela de juros compostos, com o valor total do investimento corrigido do mês zero até o mês selecionado.

Dica: procure saber mais sobre "saída formatada" na linguagem escolhida. Veja o exemplo de saída produzida usando o comando de impressão formatada `print` (python) ou `printf` (em diferentes linguagens):

```
printf("%5d %,20.2f %,20.2f %,20.2f\n", n, Jn, Jt, I );
```

```
Mês juros no mês    juros total  Investimento
  0      0.00           0.00     100.00
  1      1.00           1.00     101.00
  2     1.51            2.51         102.51
  3     2.02            4.53     104.53
```

---

5. a) Utilize laços para calcular o Mínimo Múltiplo Comum (MMC) de dois números entrados pelo usuário; b) Faça uma função que receba dois números como parâmetros e retorne o MMC ao ponto de chamada.

4.14 Processando a Informação: Cap. 4: Estruturas de Repetição (Laços) - Prática 2

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

4.14.1 Exercícios

Fontes: ref1; ref2; ref3; ref4

---

1. Desenvolver um algoritmo que leia a altura de 15 pessoas. Este programa deverá calcular e mostrar :

a. A menor altura do grupo;

b. A maior altura do grupo;

---

2. Escrever um algoritmo que leia uma quantidade desconhecida de números e conte quantos deles estão em cada um dos seguintes intervalos: [0-25], [26-50], [51-75] e [76-100]. A entrada de dados deve terminar quando for lido um número negativo.

---

3. Escrever um algoritmo que gera e escreve os números ímpares entre 100 e 200. Utilizar o método definido da lista anterior (Prática 1).

---

4. Escreva um algoritmo que leia um valor inicial a_1 e uma razão r e imprima uma sequência em P.A. contendo 10 valores. Relembrando:

$$a_n = a_1 + (n-1)r$$

---

5. Escreva um algoritmo que leia um valor inicial a_1 e uma razão q e imprima uma sequência em P.G. contendo 10 valores. Relembrando:

$$a_n = a_1 q^{n-1}$$

---

6. Escreva um algoritmo que leia um valor inicial A e imprima a sequência de valores do cálculo de A! e o seu resultado. Formatar a saída exatamente como no exemplo:

$$5! = 5X4X3X2X1 = 120$$

---

7. Foi feita uma pesquisa entre os habitantes de uma região e coletados os dados de altura e sexo (0=masc, 1=fem) das pessoas. Faça um programa que leia 50 dados diferentes e informe:

- a maior e a menor altura encontradas
- a média de altura das mulheres
- a média de altura da população
- o percentual de homens na população

---

8. Rafa tem 1,50 metro e cresce 2 centímetros por ano, enquanto Zé tem 1,10 metro e cresce 3 centímetros por ano. Construa um algoritmo que calcule e imprima quantos anos serão necessários para que Zé seja maior que Rafa.

---

9. Em uma eleição presidencial existem quatro candidatos. Os votos são informados através de códigos. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

- 1,2,3,4 = voto para os respectivos candidatos;
- 5 = voto nulo;
- 6 = voto em branco;

Elabore um algoritmo que leia o código do candidato em um voto. Calcule e escreva:

- total de votos para cada candidato;
- total de votos nulos;
- total de votos em branco;

Como finalizador do conjunto de votos, tem-se o valor 0.

---

10. Em uma eleição presidencial existem quatro candidatos. Os votos são informados através de códigos. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

- 1,2,3,4 = voto para os respectivos candidatos;
- 5 = voto nulo;
- 6 = voto em branco;

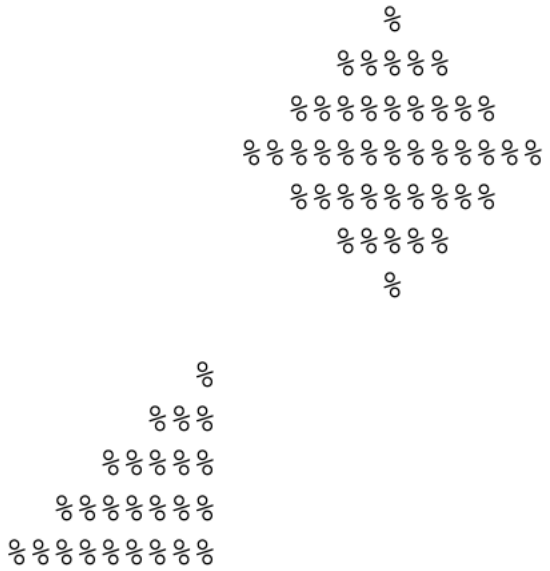
Elabore um algoritmo que leia o código do candidato em um voto. Calcule e escreva:

- total de votos para cada candidato;
- total de votos nulos;
- total de votos em branco;

Como finalizador do conjunto de votos, tem-se o valor 0.

---

11. Faça um programa que desenhe na tela losangos ou triângulos utilizando somente o caractere "%" (veja exemplos abaixo). O usuário é quem escolhe o que deve ser impresso. O usuário também deve ter a opção de escolher o tamanho (em linhas) da figura a ser desenhada.



---

12. Construa um algoritmo para o jogo da velha. Esse jogo consiste em um tabuleiro de dimensão 3x3 de valores O ou X. Os usuários devem informar a linha e a coluna que desejam preencher. A partir da terceira jogada de cada jogador é necessário verificar se houve algum ganhador. Também é possível que o resultado do jogo seja empate (nenhum jogador preencheu uma coluna, uma linha ou uma diagonal).

---

13. Faça um programa que calcule e mostre o maior divisor comum de dois números a e b, usando o algoritmo básico de Euclides (temp é uma variável inteira, temporária):

```
enquanto b for diferente de 0:
  temp = a
  a = b
  b = temp % b
```

Esse algoritmo deixará o resultado (MDC) em a, no final.

4.15 Processando a Informação: Cap. 4: Estruturas de Repetição (Laços) - Prática 3

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

4.15.1 Exercícios

Fontes: ref1; ref2

---

1. Escreva um algoritmo que solicite que o usuário entre com valores inteiros positivos quaisquer. A condição de parada é digitar um número negativo (< 0). Ao final imprima a quantidade de números digitados, o somatório dos valores digitados, e a média aritmética do somatório.

---

2. Elabore um algoritmo para fazer cálculo de potenciação. Ou seja, x^y. Exemplo: 3^4 = 3 x 3 x 3 x 3. Seu algoritmo deverá solicitar que o usuário entre com o valor da base (x) e do expoente (y) e apresentar o resultado do cálculo sem utilizar os operadores ** ou ^. Para resolver o problema utilize estrutura de repetição.

---

3. Escreva um algoritmo que calcule a média da seguinte seqüência numérica a seguir: $1/2+1/3+1/4+1/5+1/6+\cdots+1/50$. Feito isto, o algoritmo deverá apresentar uma lista contendo todos os números da seqüencia que estão acima da média calculada.

---

4. Apresente o que será impresso na tela do computador pelos algoritmos a seguir:

a) EXEMPLO

```
início
 declare J, I, X : inteiro
 J ← 100
 X ← 3
 J ← J + 40
 I ← 5 ^ X * 4
 enquanto (X >= 5) então
  J ← J - 15
  X ← X + 1
  I ← I + X - J
 fim enquanto
 escreva J, I, X
fim
```

Resposta:

J	X	I
100		
	3	
140		
		500

saída: 140, 500, 3

b)

```
início
 declare J, I, X : inteiro
 J ← 100
 X ← 3
 J ← J + 40
 I ← 5 ^ X * 4
 repita
  J ← J - 15
  X ← X + 1
  I ← I + X - J
 enquanto (X >= 5)
 escreva J, I, X
fim
```

Sua Resposta:

J	X	I

c)

```
início
 declare J, I, X : inteiro
 J ← 100
 X ← 3
 J ← J + 40
 I ← 5 ^ X * 4
 enquanto (X <= 5) faça
  J ← J - 15
  X ← X + 1
  I ← I + X - J
 fim enquanto
 escreva J, I, X
fim
```

Sua Resposta:

J	X	I

d)

```
início
 declare M, N, Y : inteiro
 M ← 10
 Y ← 1
 para N ← 1 até 3 passo 1 faça
  M ← M - 8
  Y ← Y * 3
 fim para
 escreva M, Y, N
fim
```

Sua Resposta:

M	N	Y

e)

```
início
 declare P, Q : inteiro
 declare VALOR : real
 P ← 5
 Q ← P - 8
 VALOR ← 18
 repita
  VALOR ← VALOR + (VALOR * P + Q)
  P ← P + 2
```

```
  Q ← Q + 1
 enquanto (Q < 0)
 escreva VALOR
fim
```

Sua Resposta:

P	Q	Valor

f)

```
início
 declare CONT : inteiro
 declare VALOR : real
 declare RESP : caracter
 CONT ← 0
 VALOR ← 0
 RESP ← 's'
 enquanto (RESP = 's') faça
  VALOR ← VALOR + 139
  CONT ← CONT + 1
  se (CONT > 3) então
   RESP ← 'n'
  fim se
 fim enquanto
 escreva VALOR
fim
```

Sua Resposta:

X	X	X

g)

```
início
 declare N : inteiro
 declare SOMA : real
 SOMA ← 0
 para N ← 1 até 5 passo 1 faça
  SOMA ← SOMA + 1 / N
 fim para
 escreva SOMA
fim
```

Sua Resposta:

X	X	X

h)

```
início
 declare N : inteiro
 N ← 0
 enquanto (N < 5) faça
  se (N = 0) então
   escreva ‘‘Esse número não existe: 1/0’’
  senão
  escreva 1 / N
  fim se
  N ← N + 1
 fim enquanto
fim
```

Sua Resposta:

X	X	X

---

5. A prefeitura de uma cidade fez uma pesquisa entre seus habitantes, coletando dados sobre o salário e número de filhos. A prefeitura deseja saber:

   a) média do salário da população;

   b) média do número de filhos;

   c) maior salário;

   d) percentual de pessoas com salário até R$10000,00.

O final da leitura de dados se dará com a entrada de um salário negativo. (Use o comando ENQUANTO-FAÇA)

---

6. Escreva um algoritmo que leia o código de um aluno e suas três notas. Calcule a média ponderada do aluno, considerando que o peso para a maior nota seja 4 e para as duas restantes, 3. Mostre o código do aluno, suas três notas, a média calculada e uma mensagem “APROVADO” se a média for maior ou igual a 5 e “REPROVADO” se a média for menor que 5. Repita a operação até que o código lido seja negativo.

---

7. Escrever um algoritmo que lê um conjunto não determinado de valores, um de cada vez, e escreve uma tabela com cabeçalho, que deve ser repetido a cada 20 linhas. A tabela conterá o valor lido, seu quadrado, seu cubo e sua raiz quadrada.

---

8. Escrever um algoritmo que lê um número não determinado de pares de valores m, n, todos inteiros e positivos, um par de cada vez, e calcula e escreve a soma dos n inteiros consecutivos a partir de m inclusive.

---

9. Escrever um algoritmo que lê um número não determinado de valores para m, todos inteiros e positivos, um de cada vez. Se m for par, verificar quantos divisores possui e escrever esta informação. Se m for ímpar e menor do que 10 calcular e escrever o fatorial de m. Se m for ímpar e maior ou igual a 10 calcular e escrever a soma dos inteiros de 1 até m.

---

10. Faça um algoritmo que leia uma quantidade não determinada de números positivos. Calcule a quantidade de números pares e ímpares, a média de valores pares e a média geral dos números lidos. O número que encerrará a leitura será zero.

---

11. Foi realizada uma pesquisa de algumas características físicas da população de um certa região. Foram entrevistadas 500 pessoas e coletados os seguintes dados:

a) sexo: M (masculino) e F (feminino)

b) cor dos olhos: A (azuis), V (verdes) e C (castanhos)

c) cor dos cabelos: L (louros), C (castanhos) e P (pretos)

d) idade

Deseja-se saber:

a maior idade do grupo a quantidade de indivíduos do sexo feminino, cuja idade está entre 18 e 35 anos e que tenham olhos verdes e cabelos louros.

---

12. Foi feita uma estatística nas 200 principais cidades brasileiras para coletar dados sobre acidentes de trânsito. Foram obtidos os seguintes dados:

- código da cidade
- estado (RS, SC, PR, SP, RJ, ...)
- número de veículos de passeio (em 1992)
- número de acidentes de trânsito com vítimas (em 1992)

Deseja-se saber:

a) qual o maior e o menor índice de acidentes de trânsito e a que cidades pertencem

b) qual a média de veículos nas cidades brasileiras

c) qual a média de acidentes com vítimas entre as cidades do Rio Grande do Sul.

---

13. Uma loja tem 150 clientes cadastrados e deseja mandar uma correspondência a cada um deles anunciando um bônus especial. Escreva um algoritmo que leia o nome do cliente e o valor das suas compras no ano passado e calcule um bônus de 10% se o valor das compras for menor que 500.000 e de15 %, caso contrário.

---

14. Faça um algoritmo que mostre os conceitos finais dos alunos de uma classe de 75 alunos, considerando (use o comando CASO):

a) os dados de cada aluno (número de matrícula e nota numérica final) serão fornecidos pelo usuário

b) a tabela de conceitos segue abaixo:

Nota	Conceito
de 0,0 a 4,9	D
de 5,0 a 6,9	C
de 7,0 a 8,9	B
de 9,0 a 10,0	A

5 Processando a Informação: Cap. 5: Vetores

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

5.1 Sumário

- Revisão do capítulo anterior
- Introdução
- Trabalhando com vetores
- Acessando elementos de um vetor
- Formas de percorrer um vetor
- Modularização e vetores
- Eficiência de Algortimos
- Revisão deste capítulo
- Exercícios

5.2 Revisão do capítulo anterior (Estruturas de Repetição - Laços)

- Estruturas de repetição (laços) devem ser utilizadas quando existem instruções que se repentem e podem ser de vários tipos, dependendo da linguagem utilizada, por exemplo:
  - `do-while` (não aceita em Python)
  - `while` (a maioria)
  - `for` (a maioria)
  - `repeat` (R)
- O uso de laços depende da lógica a ser implementada e da linguagem utilizada. Mas, se tiver um número fixo de iterações, geralmente se usa `for`.
- Validação de dados utilizando laços:

  - Incluir um laço para verificar o valor lido.
- Interrupção da execução em laços:
  - Depende da linguagem, algumas possibilidades:
    * `break` - interrompe o laço
    * `continue` - não executa o final do laço
    * `exit` - aborta o laço e o programa!
- Neste capítulo iremos abordar a estrutura de **Vetor**, incluindo os conceitos apresentados nos capítulos anteriores.

5.3 Introdução

- Um **vetor** em programação é formado por um **conjunto de n variáveis de um mesmo tipo** de dado (em geral), sendo n obrigatoriamente maior que zero.
- Essas variáveis (ou elementos) são identificadas e acessadas por um **nome** e um **índice**.
- Na maioria das linguagens de programação, o **índice** recebe valores de 0 (**primeiro elemento**) até $n - 1$ (**último elemento**).

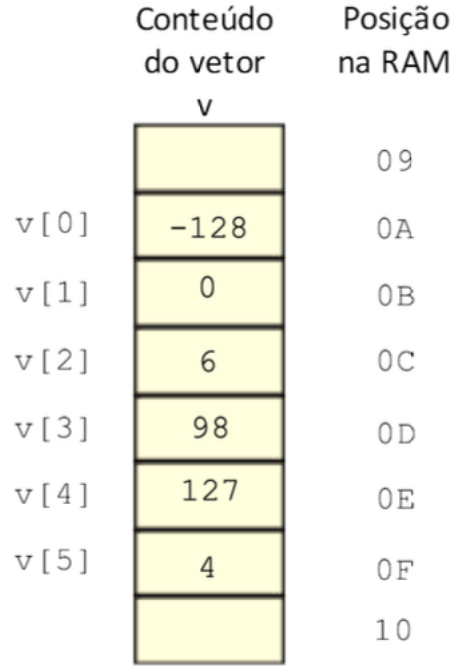
- As variáveis de um vetor são armazenadas em posições consecutivas de memória.

5.4 Trabalhando com vetores

- Quando um vetor é criado, é instanciada uma variável do tipo vetor (em algumas linguagens de programação, sendo necessário se definir um nome, o tipo de dado e o tamanho do vetor).
- Na linguagem C, por exemplo, a definição do tamanho de um vetor ocorre antes de compilar o programa.
  - Ou seja, o programador deverá informar no código a quantidade de memória a ser reservada para o vetor, através de um número inteiro positivo ou constante que o contenha, especificando o número de elementos do mesmo tipo que a memória reservada irá comportar.
  - Neste caso, uma variável não pode ser usada.
- Já na linguagem Java, é possível alocar memória em tempo de execução do programa.
  - Por exemplo, é possível criar um vetor para armazenar as notas de uma turma de alunos, onde o número de alunos é uma variável a ser lida durante a execução do programa.

- Em muitas linguagens, como Java e C, o processo de criar um vetor ocorre em dois passos distintos:
  1. primeiro se cria uma variável de referência para o vetor;
  2. em seguida se reserva a memória para um dado número de elementos do mesmo tipo.
- No exemplo da figura abaixo,
  - a criação da variável de referência de um vetor `v` define apenas a posição de memória em hexadecimal `0A`, onde será armazenado o seu primeiro elemento.
    * neste exemplo, `V[0] = -128` é o primeiro elemento do vetor.
    * a quantidade de `bytes` por elemento vai depender to tipo de dado armazenado. Neste exemplo cada elemento ocupa um `byte`.

– o segundo passo da criação de um vetor é reservar (ou alocar) memória para todos os seus elementos (dependendo da linguagem). > essa alocação de memória pode ocorrer em tempo de execução, como ocorre em Python!



- Note que a variável `v` acima agora se refere a uma posição de memória e não a um elemento de dado.
- Os elementos devem ser acessados através do índice, um por vez.
- Isso pode requerer uso de laços para leitura e impressão dos seus elementos, dependendo da linguagem de programação utilizada.
  – O índice de vetor é sempre um número inteiro.

5.5 Exemplo 01 - Ler/Escrever vetor

Pseudocódigo

```
Instanciar o vetor v1 de inteiro com tamanho 5 // ou
vetor v2 = vetor de inteiros com 100 elementos // ou ainda
vetor v3 = inteiro(10)
v1 = {4,1,10,2,3} // atribuindo valores a v1
```

Exemplo 01: Ler uma lista com `n` alunos com `RA` e `Nome` e escrever formatando a saída como no exemplo:

```
LISTA DE ALUNOS
Número RA     Nome
1      2134   Maria Campos
2      346    João Silva
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto (pode

incluir mais casos):

```
case=caso1
input=2
2345
9870
Maria Campos
João Silva
output=
LISTA DE ALUNOS
Número RA     Nota
1      2345   Maria Campos
2      9870   João Silva
```

[ ]:
```
%%writefile cap5ex01.c
#include <stdio.h>

int main(void) {

  // ENTRADA DE DADOS
  int max = 100; // considerar sempre um número grande
  int n, ra[max];     // aloca 100 ra's
  char nome[max][40]; // aloca 100 nomes de até 40 caracteres cada

  printf("Digite o numero de alunos: ");
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    printf("RA: ");
    scanf("%d", &ra[i]);
    printf("Nome: ");
    scanf("%s", nome[i]); // ATENÇÃO: NÃO ACEITA NOME COM ESPAÇOS e NÃO␣
 ↪USA &
  }
  // PROCESSAMENTO ?
  // SAÍDA
  printf("LISTA DE ALUNOS\nNúmero\t RA\t Nome\n");
  for (int i = 0; i < n; i++) {
    printf("%d\t %d\t %s\n", i + 1, ra[i], nome[i]);
  }
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex01.c -o output2
./output2
```

- As *strings* em C têm um último caracter `'\0'`. Assim, no exemplo a seguir `s1=s2` tem três carateres cada:
  - `char s1[] = "oi";` ou

  - `char s2[] = {'o','i','\0'};`
- Para ler uma *string* com espaços [ref]:

[ ]:
```
%%writefile cap5ex01teste.c
#include <stdio.h>
#include <string.h>

int main() {
   char s[40];
   printf("digite algo:\n");
   fgets(s, 40, stdin);
   printf("saida: \"%s\" tamanho: %ld\n", s, strlen(s));
   s[strlen(s)-1] = '\0'; // substituir \n por \0
   printf("saida: \"%s\" tamanho: %ld\n", s, strlen(s));
   return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex01teste.c -o cap5ex01teste
./cap5ex01teste
```

Algumas funções da biblioteca padrão **string.h**:

função	descrição
strcopy(s1,s2)	copia s1 em s2
strcat(s1,s2)	copia s2 no final de s1
strlen(s)	tamanho de s
strcmp(s1,s2)	0 se s1=s2; negativo se s1<s2; positivo se s1>s2
strchr(s,ch)	ponteiro para a primeira ocorrência de ch em s
strstr(s1,s2)	ponteiro para a primeira ocorrência de s2 em s1

5.6 Formas de Percorrer um Vetor

5.6.1 Percorrer um vetor com o laço `para`

- Como um vetor, após criado (alocado e com elementos), tem tamanho fixo (sempre com número de elementos `n>0`), geralmente é indicado usar estruturas de repetição do tipo `para` (`for`), de forma a percorrer todos os elementos do vetor
  - assim tornando o código genérico para um vetor de qualquer tamanho `n`.
- Além disso, é natural percorrer (ou varrer) o vetor da posição `i=0` até a posição `i<n`
  - O índice `i` assume automaticamente os valores `0`, `1`, `2`, até `n-1`, em cada iteração do laço.
  - Em algumas linguagens, como Matlab e R, o índice assume valores entre `1` até `n`.
- No exemplo a seguir em pseudocódigo, um vetor `v` é criado com 6 posições e o laço `para` inicializa todos os seus elementos com o valor 0.

- A forma natural de percorrer os elementos de um vertou (ou **varrendo** o vetor) é na ordem **raster**, do primeiro até o último elemento.
- A vantagem de se usar a estrutura `para` para varrer um vertor é permitir embutir na própria sintaxe (linha) da instrução:
  - declaração e inicialização do índice,
  - incremento e
  - condição de saída do laço.

Pseudocódigo

```
Instanciar um vetor v de inteiro com 6 elementos (n=6)
para cada indice i, de i=0; até i<n; passo i=i+1 faça
    v[i] = 0
```

5.6.2 Percorrer um vetor com o laço `enquanto`

- Se o programador preferir, ou o problema a ser resolvido exigir, o vetor pode alternativamente ser percorrido utilizando-se uma estrutura de repetição do tipo `enquanto`:

Pseudocódigo

```
n=6
vetor v de inteiros com n elementos
inteiro i=0
enquanto i<n faça {
    v[i] = 0
    i=i+1
}
```

- O código usando a estrutura de repetição `enquanto` produz o mesmo resultado que usando com `para`, porém, usando mais instruções:
  - a criação do contador `i`
  - o incremento `i=i+1`.
- Essas duas instruções ficam encapsuladas na própria instrução `para`.

5.6.3 Outras formas de percorrer um vetor

- Dependendo do problema a ser tratado, existem várias formas de se varrer um vetor usando estruturas de repetição para acessar cada elemento.
- Por exemplo, é possível percorrer o vetor do último elemento para o primeiro (essa varradora é chamada de **anti-raster**):

Pseudocódigo: percorrer um vetor usando para na ordem inversa (anti-raster).

```
Instanciar um vetor v de inteiro com 6 elementos (n=6)
para cada indice i, de i=n-1; até i<=0; passo i=i-1 faça
    v[i] = 0
```

- Também, é possível percorrer somente alguns elementos do vetor, como para acessar os elementos que estão nos índices pares ou ímpares, diferenciadamente:

Pseudocódigo: percorrer um vetor usando `para`, com passo 2.

```
vetor v de inteiro com 6 elementos
n=6
para cada indice i, de i=0; até i<n-1; passo i=i+2 faça {
    v[i] = 0
    v[i+1] = 1
}
```

5.7 Modularização e Vetores

- Como uma boa prática de programação, além de comentar os códigos e organizar com tabulação, como apresentado nos exemplos deste livro,
  - também é recomendável modularizar o código usando métodos, como apresentado no Capítulo 2.
- Todo sistema computadorizado de informação possui três partes bem definidas, agrupadas em
  - módulo(s) de entrada,
  - módulo(s) de processamento e
  - módulo(s) de saída.
- Ao manipular informações armazenadas em vetores, é natural criar também pelo menos três módulos ou métodos:
  - `leiaVetor`,
  - `processaVetor` e
  - `escrevaVetor`,

  satisfazendo essa definição de sistema de informação.

  Para a reutilização de código, os módulos `leiaVetor` e `escrevaVetor` poderão ser muito reutilizados para resolver outros problemas de manipulação de vetores.

Pseudocódigo: método para ler um vetor de inteiro tamanho n.

```
método leiaVetor(inteiro n): retorna vetor de inteiro v[]
    vetor v de inteiros com n elementos
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        v[i] = leia("Entre com o elemento " + i + ":");
    retorne v
```

Pseudocódigo: método para escrever um vetor de inteiro tamanho n.

```
método escrevaVetor(inteiro v[], inteiro n):
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        escreva(" " + v[i]);
```

Pseudocódigo: aplicação de vetor usando módulos.

```
// Instâncias e Atribuições
inteiro n = leia("Digite o tamanho do vetor:");
```

```
// ENTRADA
inteiro v1[] = leiaVetor(n)

// PROCESSAMENTO
// ?

// SAÍDA
escrevaVetor(v2, n)
```

5.8 Exemplo 02 - Aplicação simples 1 usando vetor

Considere um algoritmo para ler a quantidade `n` de alunos de uma turma. Ler uma lista com `n` RA's de alunos. Em seguida, ler também uma lista com `n` notas de alunos. Como saída do algoritmo, escrever a seguinte saída.

```
LISTA DE ALUNOS
Número RA     Nota
1      2134   9
2      346    7
```

Utilizar os métodos `leiaVetor` e `escrevaVetor`, se necessários.

Pseudocódigo

```
// ENTRADAS:
// instâncias e atribuições
real media, somador = 0
inteiro contador = 0

// leitura do número de alunos = tamanho do vetor
inteiro n = leia("Digite o número de alunos:");

inteiro ras[] = leiaVetor(n)
inteiro notas[] = leiaVetor(n)

// PROCESSAMENTO
// ?

// SAÍDAS:
escreva("LISTA DE ALUNOS")
escreva("Número RA     Nota")
para cada indice i, de i=0; até i<n; passo i=i+1 faça
    escreva(i+1,"\t",ras[i],"\t",notas[i]
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 02, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=2
```

```
3456
2345
5
2
output=
LISTA DE ALUNOS
Número   RA Nota
1    3456    5
2    2345    2
```

[ ]:
```
%%writefile cap5ex02.c
#include <stdio.h>

int main(void) {

  // ENTRADA DE DADOS
  int max = 100; // número máximo de alunos
  int n, ras[max], notas[max];   // variaveis de referência ras e notas

  printf("Digite o numero de alunos: \n");
  scanf("%d", &n);

  printf("RAs: \n");
  for (int i = 0; i < n; i++) {
    printf("RA %d: \n", i + 1);
    scanf("%d", &ras[i]);
  }

  printf("Notas: \n");
  for (int i = 0; i < n; i++) {
    printf("Nota %d: \n", i + 1);
    scanf("%d", &notas[i]);
  }

  // PROCESSAMENTO ?
  // SAÍDA
  printf("LISTA DE ALUNOS\nNúmero\t RA\t Nota\n");
  for (int i = 0; i < n; i++) {
    printf("%d\t %d\t %d\n", i + 1, ras[i], notas[i]);
  }
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex02.c -o output2
./output2
```

Existem várias formas de usar vetor em métodos:

- como argumento:
  - `void metodo (int *v, int n) {...}`
  - `void metodo (int v[], int n) {...}`
  - `void metodo (int v[max], int n) {...}`
- como retorno:
  - `int * metodo (int n) {...}` - alocar vetor dentro do método, alocação dinâmica de memória, com `malloc` ou `calloc`.
  - não esquecer de liberar a memórica com `free`.

[ ]:
```
%%writefile cap5ex02teste1.c
#include <stdio.h>

#define MAX_ALUNOS 20 // número máximo de alunos

void leiaVetor(int *v, int n) {
  for (int i = 0; i < n; i++)
    scanf("%d", &v[i]);
}

int main(void) {

  // ENTRADA DE DADOS
  int n, ras[MAX_ALUNOS], notas[MAX_ALUNOS];   // variaveis de␣
 ↪referência ras e notas

  printf("Digite o numero de alunos: \n");
  scanf("%d", &n);

  printf("RAs: \n");
  leiaVetor(ras, n);

  printf("Notas: \n");
  leiaVetor(notas, n);

  // PROCESSAMENTO ?
  // SAÍDA
  printf("LISTA DE ALUNOS\nNúmero\t RA\t Nota\n");
  for (int i = 0; i < n; i++)
    printf("%d\t %d\t %d\n", i + 1, ras[i], notas[i]);

  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex02teste1.c -o output2
./output2
```

[ ]:
```
%%writefile cap5ex02teste2.c
#include <stdio.h>
#include <stdlib.h>  // malloc e free

int * leiaVetor(int n) {
  int *v= malloc(sizeof(n));  // ALOCAÇÃO DINÂMICA
  for (int i = 0; i < n; i++) {
    scanf("%d", &v[i]);
  }
  return v;
}

int main(void) {

  // ENTRADA DE DADOS
  int n, *ras, *notas;   // variaveis de referência ras e notas

  printf("Digite o numero de alunos: ");
  scanf("%d", &n);

  printf("RAs: ");
  ras = leiaVetor(n);

  printf("Notas: ");
  notas = leiaVetor(n);

  // PROCESSAMENTO ?
  // SAÍDA
  printf("LISTA DE ALUNOS\nNúmero\t RA\t Nota\n");
  for (int i = 0; i < n; i++) {
      printf("%d\t %d\t %d\n",i+1, ras[i],notas[i]);
  }
  free(ras);   // LIBERAR MEMÓRIA ALOCADA COM malloc
  free(notas);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex02teste2.c -o output2
./output2
```

5.9 Exemplo 03 - Aplicação simples 2 usando vetor

Considere um algoritmo para ler a quantidade `n` de alunos de uma turma. Ler também uma lista com `n` notas de alunos. Como saída do algoritmo, escrever a média da turma e quantos alunos ficaram acima da média.

Pseudocódigo

```
// ENTRADAS:
// instâncias e atribuições
real media, somador = 0
inteiro contador = 0

// leitura do número de alunos = tamanho do vetor
inteiro n = leia("Digite o número de alunos:");

inteiro ras[] = leiaVetor(n)
real notas[] = leiaVetor(n)

// PROCESSAMENTO: soma, média e contador
para cada indice i, de i=0; até i<n; passo i=i+1 faça
    somador = somador + notas[i] // soma
media = somador / n   // média
para cada indice i, de i=0; até i<n; passo i=i+1 faça
    se ( notas[i] >= media ) // conta alunos>=media
        contador = contador + 1

// SAÍDAS:
escreva("Média da turma=" + media) // Saída
escreva("Alunos acima da média: " + contador)
escreva("LISTA DE ALUNOS ACIMA DA MÉDIA")
escreva("RA\t Nota")
para cada indice i, de i=0; até i<n; passo i=i+1 faça
  se (nota[i] >= media)
    escreva(ras[i],"\t",notas[i]
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 03, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=2
233245
234534
9
4
output=
Média da turma = 6.5
Número de alunos acima da média = 1
LISTA DE ALUNOS ACIMA DA MÉDIA
RA      Nota
233245  9
```

[ ]:
```
%%writefile cap5ex03.c
#include <stdio.h>
#include <stdlib.h>
```

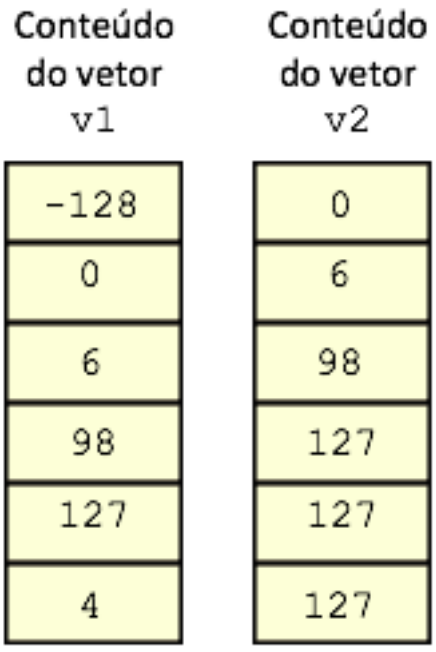
```c
int * leiaVetor(int n) {
  int *v= malloc(sizeof(n));
  for (int i = 0; i < n; i++) {
    scanf("%d", &v[i]);
  }
  return v;
}

int main(void) {

  // ENTRADA DE DADOS
  int n, *ras, *notas;   // variaveis de referência ras e notas
  float media,somador = 0.0;
  int contador = 0;

  printf("Digite o numero de alunos: ");
  scanf("%d", &n);

  printf("RAs: ");
  ras = leiaVetor(n);

  printf("Notas: ");
  notas = leiaVetor(n);

  // PROCESSAMENTO: soma, média e contador
  for (int i = 0; i < n; i++) {
      somador = somador + notas[i];   // soma
  }
  media = (float) somador / n;             // média

  for (int i = 0; i < n; i++) {
      if (notas[i] >= media) {        // conta alunos>=media
          contador = contador + 1;
      }
  }
  // SAÍDA DE DADOS
  printf("Média da turma = %.1f\n",media);
  printf("Número de alunos acima da média = %d\n",contador);
  printf("LISTA DE ALUNOS ACIMA DA MÉDIA\nRA\t Nota\n");
  for (int i = 0; i < n; i++) {
    if (notas[i] >= media) {        // conta alunos>=media
      printf("%d\t %d\n",ras[i],notas[i]);
    }
  }
  free(ras);  // liberar memória alocado com malloc
  free(notas);
```

```
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex03.c -o output2
./output2
```

5.10 Exemplo 04 - Aplicação “dilata” vetor

- O programador deve tomar cuidado ao acessar os elementos de um vetor para **não ultrapassar os seus limites alocados previamente**, ou seja,
  - para um vetor `v` de tamanho `n`, índices `i` com `i<0` ou `i>=n` não existem.
- O exemplo a seguir ilustra este problema.
- Considere o problema de “**dilatar**” um vetor.
  - O objetivo é criar um vetor `v1` de inteiros com `n` posições e um outro vetor `v2` de inteiros também com `n` posições.
  - Cada posição `i` de `v2` armazena o cálculo do **máximo** entre cada elemento `i` em `v1` e seus vizinhos:
    1. à esquerda `v1[i-1]`,
    2. do próprio elemento `v1[i]` e
    3. do seu vizinho à direita `v1[i+1]`.
- Veja uma ilustração da operação de dilatação na Figura abaixo, onde `v1` é o vetor de entrada e `v2` é o vetor de saída, contendo a “dilatação” de `v1`, seguido do código para resolver este problema proposto.



Pseudocódigo

```
método escrevaVetor(inteiro v[], inteiro n):
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        escreva(" " + v[i]);

método leiaVetor(inteiro n): retorna vetor de inteiro v[]
    vetor v de inteiros com n elementos
    para cada índice i, de i=0; até i<n; passo i=i+1 faça
        v[i] = leia("Entre com o elemento " + i + ":");
    retorne v

// Inicializações
inteiro max=0
// ENTRADA
inteiro n = leia("Digite o tamanho do vetor:");
vetores v1 e v2 de inteiros com n elementos

inteiro v1[] = leiaVetor(n)

// PROCESSAMENTO
para cada índice i, de i=0; até i<n; passo i=i+1 faça
    max = v[i]
    se i-1 >= 0 e max < v1[i-1] faça
        max = v1[i-1]
    se i+1 < n e max < v1[i+1] faça
        max = v1[i+1]
    v2[i] = max

// SAÍDA
escrevaVetor(v2)
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 04, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=6
-128
0
6
98
127
4
output=v2:
0
6
98
127
```

```
127
127
```

[ ]:
```
%%writefile cap5ex04.c
#include <stdio.h>
#include <stdlib.h>  // malloc e free

int * leiaVetor(int n) {
  int *v = malloc(n*sizeof(int));
  for (int i = 0; i < n; i++) {
    scanf("%d", &v[i]);
  }
  return v;
}
void escrevaVetor(int *v, int n) {
  for (int i = 0; i < n; i++) {
    printf("%d\n", v[i]);
  }
}
int main(void) {
  // ENTRADA DE DADOS
  int n, *v1;   // variaveis de referência v1
  printf("Digite o tamanho do vetor: ");
  scanf("%d", &n);
  int *v2 = malloc(100*sizeof(int));
  printf("Digite os elementos: ");
  v1 = leiaVetor(n);
  // PROCESSAMENTO
  for (int i = 0; i < n; i++) {
    int max = v1[i];
    if (i-1 >= 0 && max < v1[i-1])
        max = v1[i-1];
    if (i+1 < n && max < v1[i+1])
        max = v1[i+1];
    v2[i] = max;
  }
  // SAÍDA:
  printf("\nv2:\n");
  escrevaVetor(v2,n);
  free(v1); // liberar memória alocado com malloc
  free(v2);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap5ex04.c -o output2
./output2
```

5.11 Eficiência de Algoritmo

A eficiência de um código (ou melhor, de um algoritmo) se mede analisando (ou contando) o número de instruções executadas.

Este tipo de análise de complexidade é útil quando temos uma grande quantidade de dados a serem processados.

Assim, um algoritmo mais eficiente (com menos passos) resolve um mesmo problema mais rápido. Imagine, por exemplo, que uma simulação física em tempo real é bem melhor que uma que leva várias horas para gerar resultados. Apesar de ser desejável medir a eficiência de um algoritmo considerando a medida de tempo, esta medida depende do sistema utilizado (hardware).

Para resolver isto, a análise da complexidade de algoritmo se mede através da contagem do número de instruções executadas.

Para simplificar a contagem, podemos considerar apenas algumas instruções, como a quantidade de algumas condições lógicas.

5.11.1 Busca

Por exemplo, considerando um algoritmo para buscar um elemento x em um vetor de inteiros com n elementos, temos, no melhor caso, apenas uma instrução, considerando que o elemento x está na primeira posição do vetor.

Neste caso dizemos que o algoritmo de busca possui uma complexidade assintótica1 de melhor caso $\Omega(1)$, ou complexidade constante (notação Bachmann–Landau, assintótica ou, como é mais conhecida, Big-O notation).

Verifique o teste de mesa a seguir considerando um vetor v com 5 elementos, contendo valores de 1 a 5, e considerando também que $x = 1$.

	Função Busca(int vet, int n, int x)	i	v[i]	Busca(v,5,1)
1	para i=0; i < n; i++	0		
2	_ se v[i]==x		1	
3	_ _ retorne i			0
4	retorne -1			
	Valores finais			0

Por outro lado, este algoritmo de busca tem no pior caso n instruções de comparação, ou seja, se $x = 5$, a instrução na linha 2 será executada 5 vezes. Neste caso, dizemos que o algoritmo de busca tem complexidade assintótica de pior caso $O(n)$, ou complexidade linear.

Quando $\Omega(n) = O(n)$, então dizemos que é um algoritmo ótimo, ou $\Theta(n)$ (da ordem de n).

Por exemplo, considere o algoritmo buscaMaior que encontra o maior elemento de um vetor de tamanho n. Neste caso, o algoritmo buscaMaior tem complexidade assintótica $\Theta(n)$.

	Função buscaMaior(int vet, int n)	Número de instruções
1	int maior = v[0]	1
2	para i=1; i< n; i++	n-1
3	_ se maior < v[i]	n-1
4	_ _ maior = v[i]	< n-1
5	retorne maior	1
	Complexidade	$f(n) <= 3n - 1 = \Theta(n)$

A contagem do número de instruções pode ser realizada de forma simplificada somando todas as instruções que aparecem na última coluna da tabela anterior, assim $f(n) <= 1 + n - 1 + n - 1 + n - 1 + 1 = 2 + 3(n - 1) = 3n - 1 <= kn$, para algum $k >= 3$. Ou simplesmente, $f(n) = O(n)$. Como para este algoritmo de busca do maior elemento, o melhor caso também é linear, ou seja, $\Omega(n)$. Podemos afirmar então que este algoritmo de busca é ótimo, ou seja, $\Theta(n)$.

5.11.2 Ordenação

Algoritmos muito estudados em análise assintótica são os de ordenação, para deixar os valores de um vetor com n elementos em ordem crescente ou decrescente. Considere o seguinte algoritmo de ordenação, chamado **Bubble Sort**.

Veja uma simulação do algoritmo **Bubble Sort** com lego em: https://youtu.be/MtcrEhrt_K0.

Veja a comparação de vários algoritmos de ordenação em: https://youtu.be/ZZuD6iUe3Pc.

Ver também:

- https://pt.wikipedia.org/wiki/Bubble_sort
- https://pt.wikipedia.org/wiki/Selection_sort
- https://pt.wikipedia.org/wiki/Insertion_sort

	Função ordena(int vet, int n)	Número de instruções
1	para i=0; i<n; i++	n
2	_ para j=0; j<n; j++	n*n
3	_ _ se v[i] > v[j]	n*n
4	_ _ _ int aux = v[i]	< n*n
5	_ _ _ v[i] = v[j]	< n*n
6	_ _ _ v[j] = aux	< n*n
7	retorne v	1
	Complexidade	$f(n) = n * n = O(n^2)$

A complexidade assintótica de algoritmo considera o seu comportamento com um valor grande de dados a serem processados (n grande) e é abordado com detalhes em literaturas mais avançadas de programação.

Uma versão um pouco melhor deste algoritmo bobble sort é apresentada a seguir. As linhas 2 até 6 são executadas seguindo a soma de uma progressão aritmética: $f(n) = n+n-1+n-2+...+1 = n*(a_1+a_n)/2 = n*(1+n)/2 = n/2+n*n/2 <= n^2/2 <= n^2$.

Portanto, apesar de esta versão ser um pouco melhor que a versão anterior, ainda tem complexidade assintótica quadrática, ou $O(n^2)$.

Existem vários algoritmos de ordenação com diferentes ordens de complexidade, porém é possível se provar matematicamente que o mais eficiente entre eles não poderá ser melhor que $\Omega(n * log\ n)$.

	Função ordena2(int vet, int n)	Número de instruções
1	para i=0; i < n-1; i++	n-1
2	_ para j=i+1; j < n; j++	n+n-1+n-2+...+1
3	_ _ se v[i] > v[j]	n+n-1+n-2+...+1
4	_ _ _ int aux = v[i]	< n+n-1+n-2+...+1
5	_ _ _ v[i] = v[j]	< n+n-1+n-2+...+1
6	_ _ _ v[j] = aux	< n+n-1+n-2+...+1
7	retorne v	1
	** Complexidade **	$f(n) = n * n/2 = O(n^2)$

5.12 Exercícios

Ver notebook Colab nos arquivos `cap5.partX.lab.*.ipynb` (`X` ∈ `[2,3]` e `*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, organizadas em subpastas contidas de `"gen"`, na pasta do Google Drive colabs.

5.13 Atividades no Moodle+VPL

Algumas atividades no Moodle+VPL pedem como entradas vetores de inteiros (ou reais), **armazenados em uma única linha**. Exemplo de entrada a ser lida:

5.13.1 Entrada de Dados (cada linha contem um texto ou *string* incluindo os elementos do vetor e vários espaços “”):

```
7 3 7 9 7 7 0 9 8 4 8 9 0 1 7 8 4 1 1 0
2 1 9 4 3 6 0 9 8 4 2 8 0 6 7 3 2 4 5 9
```

Para não ter que incluir várias entradas inteiras, uma por linha, uma solução é fazer um método de leitura, passando como argumento um texto (*string*) referente a cada linha. Esse método deve retornar o vetor.

[ ]:
```
%%writefile str2int.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int MAX=100, tamanho = 0;
void leiaVetor(int v[MAX], char string[]) {
   char * token = strtok(string, " ");
   while( token != NULL ) {
```

```
      v[tamanho++] = atoi(token); // converte texto para inteiro
      token = strtok(NULL, " ");
   }
}
int main() {
   char string[] = "7 3 7 9 7 7 0 9 8 4 8 9 0 1    7 8 4 1 1 0    ";
   int v[MAX];
   leiaVetor(v, string);
   for (int i=0;i<tamanho;i++ ){
     printf("%d ",v[i]);
   }
   return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 str2int.c -o output2
./output2
```

[ ]:
```
%%writefile str2int.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int tamanho = 20;
// método para ler um vetor de inteiros digitado em uma linha,␣
 ↪separados por espaços
int* leiaVetor(int n) {
  int* v = malloc(n * sizeof(int)); // aloca um vetor de n inteiros
  char texto[512]; // espaço para todos os números como um texto
  scanf("%[^\n]", texto); // lê todos os números como um texto
  char* token = strtok(texto, " "); // lê até o primeiro espaço na␣
 ↪variável token
  int i = 0;
  while (token != NULL && i < n) {
    v[i++] = atoi(token); // guarda o valor convertido em numero no␣
 ↪vetor e avança
    token = strtok(NULL, " "); // lê até ao próximo espaço a partir de␣
 ↪onde terminou
  }
  return v;
}
int main() {
   //char string[] = "7 3 7 9 7 7 0 9 8 4 8 9 0 1    7 8 4 1 1 0    ";
   int * v = leiaVetor(tamanho);
   for (int i=0;i<tamanho;i++ ){
     printf("%d ",v[i]);
   }
   free(v); // libera memória alocada com malloc
```

```
    return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 str2int.c -o output2
./output2
```

5.13.2 Entrada de Dados (a linha contem um texto ou *string*):

```
A UFABC completou 15 anos!
```

Como contar as vogais de um texto?

5.14 Revisão deste capítulo de Vetores

- Introdução > Vetores são estruturas para armazenar vários elementos de um mesmo tipo de dados em uma única variável.
- Trabalhando com vetores > Cada linguagem possui uma sintaxe própria para declarar e alocar vetores.
- Acessando elementos de um vetor > ATENÇÃO para não acessar uma posição do vetor não reservada/alocada, geralmente `<0` e `>=n`.
- Formas de percorrer um vetor > É possível varrer um vetor na forma ***raster*** e ***anti-raster***, também usando diferentes passos, mas geralmente é passo=1.
- Modularização e vetores > Muito útil usar principalmente os módulos de `leiaVetor` e `escrevaVetor`, podendo ser reaproveitados em vários códigos.
- Eficiência de Algortimos > Busca

  Ordenação
- Exercícios
- Revisão deste capítulo de Vetores

5.15 Processando a Informação: Cap. 5: Vetores - Prática 1

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

5.15.1 Exercícios

[ref.: http://tiagodemelo.info/livros/logica/node4.html]

---

1. Criar um vetor para armazenar as notas de uma turma com n alunos e imprimir a maior e a menor nota da turma.

---

2. Criar um vetor de inteiros com n elementos. Achar o menor valor e mostrar, junto com o número de vezes ele ocorre no vetor.

---

3. Criar um vetor `v1` de inteiros com `n` elementos com valores de `0` até `9`. Criar um outro vetor `v2`, onde os elementos recebam a quantidade de ocorrências dos valores de `0` a `10`, armazenando nas posições `0` a `10` de `v2`.

---

4. Criar um vetor de inteiros com n elementos e uma função que receba um vetor e duas posições `i` e `j`, efetuando a troca dos valores das posições `i` e `j` no vetor. Verificar na função se `0<=i<n` e `0<=j<n` antes de realizar a troca.

---

5. Criar um vetor de inteiros com n elementos e ordenar os seus valores.

Ver simulações de vários algoritmos de ordenação: link

5.16 Processando a Informação: Cap. 5: Vetores - Prática 2

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

5.16.1 Exercícios

---

1. Criar um vetor de entrada com `n` posições com valores inteiros positivos e como saída criar um outro vetor também com `n` posições, onde a cada posição `i` seja atribuído a cálculo do mínimo do seu vizinho de `v1` à esquerda `i-1`, do próprio elemento `i` e do seu vizinho à direita `i+1`.

---

2. Criar um vetor com `n` posições com valores inteiros positivos e, como saída, criar um outro vetor também com `n` posições, onde em cada posição `i` seja atribuído a cálculo dos mínimos dos seus vizinhos de `v1` à esquerda `i-2` e `i-1`, do próprio elemento `i` e dos seus vizinhos à direita `i+1` e `i+2`. Generalize este código para os `m` vizinhos à esquerda e à direita.

---

3. O MMC (Mínimo Múltiplo Comum) de dois ou mais números inteiros é o menor múltiplo inteiro positivo comum a todos eles. Fazer uma função chamada MMC que recebe um vetor de números inteiros e retorna o MMC de todos. Veja um exemplo abaixo para calcular o MMC de 12 e 15:

a	b	/
12	15	2
6	15	2
3	15	3

a	b	/
1	5	5
1	1	60

$$MMC = 60 = 2 * 2 * 3 * 5$$

---

4. Criar um vetor de inteiros com n elementos. Inverter este vetor sem usar vetor auxiliar.

---

5. Criar dois vetores de inteiros com n elementos cada. Calcular o produto escalar entre eles.

5.17 Processando a Informação: Cap. 5: Vetores - Prática 3

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

5.17.1 Exercícios

Fontes: ref1, ref2, ref3, ref4, ref5

---

1. Dado um vetor qualquer com n números inteiros entre 0 e 9, faça um módulo que retorna o número que mais se repete.

---

2. Escreva um módulo que retira todos os números repetidos das primeiras N posições de um vetor em ordem crescente, colocando-os em ordem crescente no final do vetor. Exemplo: Para o vetor {1,2,2,3,3,4}, a solução é {1,2,3,4,2,3}.

---

3. Escreva um módulo que recebe um vetor lógico de 10 posições e oferece como resultado o produto da operação (((vet[0] E vet[1]) OU vet[2]) E vet[3]) ... e assim por diante.

---

4. Faça um programa que leia 30 valores do tipo inteiro e armazene-os em um vetor. A seguir, o programa deverá informar (1) todos os números pares que existem no vetor; (2) o menor e o maior valor existente no vetor; (3) quantos dos valores do vetor são maiores que a média desses valores.

---

5. Escreva um algoritmo que permita a leitura dos nomes de 10 pessoas e armaze os nomes lidos em um vetor. Após isto, o algoritmo deve permitir a leitura de mais 1 nome qualquer de pessoa e depois escrever a mensagem ACHEI, se o nome estiver entre os 10 nomes lidos anteriormente (guardados no vetor), ou NÃO ACHEI caso contrário.

---

6. Escreva um algoritmo que permita a leitura das notas de uma turma de 20 alunos. Calcular a média da turma e contar quantos alunos obtiveram nota acima desta média calculada. Escrever a média da turma e o resultado da contagem.

---

7. Ler um vetor Q de 20 posições (aceitar somente números positivos). Escrever a seguir o valor do maior (e menor) elemento de Q e a respectiva posição que ele ocupa no vetor.

---

8. Faça um algoritmo para ler um valor N qualquer (que será o tamanho dos vetores). Após, ler dois vetores A e B (de tamanho N cada um) e depois armazenar em um terceiro vetor Soma a soma dos elementos do vetor A com os do vetor B (respeitando as mesmas posições) e escrever o vetor Soma.

---

9. Faça um algoritmo para ler e armazenar em um vetor a temperatura média de todos os dias do ano. Calcular e escrever:

- Menor temperatura do ano
- Maior temperatura do ano
- Temperatura média anual
- O número de dias no ano em que a temperatura foi inferior a média anual

---

10. Faça um algoritmo para ler 10 números e armazenar em um vetor. Após isto, o algoritmo deve ordenar os números no vetor em ordem crescente. Depois de ordenar os elementos do vetor em ordem crescente, deve ser lido mais um número qualquer e inserir esse novo número na posição correta, ou seja, mantendo a ordem crescente do vetor.

---

11. Faça um algoritmo para ler um vetor de 20 números. Após isto, deverá ser lido mais um número qualquer e verificar se esse número existe no vetor ou não. Se existir, o algoritmo deve gerar um novo vetor sem esse número. (Considere que não haverão números repetidos no vetor).

---

12. Faça um algoritmo para ler dois vetores V1 e V2 de 15 números cada. Calcular e escrever a quantidade de vezes que V1 e V2 possuem os mesmos números e nas mesmas posições (também, alterar para não ter essa última restrição).

---

13. Faça um programa que leia 2 vetores com 10 elementos cada. Considerando cada vetor como sendo um conjunto, crie um terceiro vetor, que seja a união dos dois primeiros, e um quarto, que seja a intersecção entre os dois primeiros.

---

14. Dado um vetor com números ordenados de forma não decrescente, faça uma função que imprime somente os números que não sejam repetidos.

---

15. Faça uma função que recebe dois vetores de inteiros, com qualquer número de elementos cada. Ela deve imprimir todos os valores presentes nos dois vetores. Ex: se v1={19, 5, 2, 6} e v2={5, 0, 9, 4, 18, 56} deverá ser impresso somente o valor 5.

---

16. Faça um programa que dado o vetor unidimensional [2; 4; 35; 50; 23; 17; 9; 12; 27; 5] retorne:

- maior valor
- média dos valores
- os valores dispostos em ordem crescente
- sub conjunto de valores primos que está contido no vetor

---

17. Faça um programa que:

- leia 7 valores inteiros e os armazene em um vetor. Listar o vetor com as referidas-posições de armazenamento de cada valor.
- ofereça uma função de pesquisa onde dado um valor inteiro qualquer de entrada retornar a posição deste valor dentro do vetor, e caso este valor não esteja presente no vetor retornar –1.

- ofereça uma função que troque os valores contido no vetor pela seguinte política: cada elemento i dentro do vetor será substituído pela soma de todos os (i-1) elementos mais o elemento i. Por exemplo, dado um vetor [1; 2; 3; 4; 5] após a aplicação da função teríamos esse vetor preenchido com os seguintes valores [1;3; 6; 10; 15].

---

18. Escrever um programa para ler um vetor de 25 elementos do tipo inteiro e que, apósos valores serem lidos, verifique se existem números repetidos dentro do vetor. Caso exista, deverão ser informados quais são estes números e quantas vezes eles foram repetidos.

---

19. Escreva um programa em C para ler um vetor de 10 elementos inteiros. Excluir o1 o elemento do vetor deslocando os elementos subsequentes de uma posição parao inicio. Imprimir o vetor após a retirada do primeiro elemento.

---

20. Escreva um programa em C para ler um vetor X de 10 elementos e um valor P(aceitar apenas valores entre 0 e 9) que representa a posição de um elemento dentrodo vetor X. Imprimir o valor do elemento que ocupa a posição informada. Logo após excluir esse elemento do vetor fazendo com que os elementos subseqüentes (sehouverem) sejam deslocados de 1 posição para o inicio. Imprimir o vetor X após a exclusão ter sido executada.

---

21. Faça um programa que receba uma lista com nomes de alunos, as notas de cada aluno e a nota mínima para aprovação na disciplina. O aluno é considerado aprovadose a média de suas notas forma maior ou igual a nota mínima para aprovação. O programa deve informar uma lista de alunos aprovados e outra de alunos reprovados.

---

22. Escreva um programa para ler n palavras. A seguir imprimir as palavras em ordem alfabética.

---

23. Escreva um programa para ler uma frase e contar o número de ocorrências de cada uma das 15 primeiras letras do alfabeto. Imprimir as contagens.

---

24. Escreva um programa para ler uma frase e uma letra. A seguir retirar da frase, todas as letras iguais a informada. Imprimir a frase modificada.

---

25. Escreva um programa para ler uma frase e contar o número de palavras existentes na frase. Considere palavra um conjunto qualquer de caracteres separados por um conjunto qualquer de espaços em branco.

---

26. Escreva um programa que leia n números inteiros no intervalo [0,50] e os armazeneem um vetor estaticamente alocado com 100 posições. Preencha um segundo vetor, também alocado estaticamente, apenas com os números ímpares do primeiro vetor. Imprima os dois vetores, 10 elementos por linha.

---

27. Leia 10 números inteiros e armazene em um vetor v. Crie dois novos vetores v1 e v2. Copie os valores ímpares de v para v1, e os valores pares de v para v2. Note que cada um dos vetores v1 e v2 têm no máximo 10 elementos, mas nem todos os elementos são utilizados. No final escreva os elementos UTILIZADOS de v1 e v2.

---

28. Faça um programa para ler 10 números DIFERENTES a serem armazenados em um vetor. Os dados deverão ser armazenados no vetor na ordem que forem sendo lidos, sendo que caso o usuário digite um número que já foi digitado anteriormente, o programadeverá pedir para ele digitar outro número. Note que cada valor digitado pelo usuário deve ser pesquisado no vetor, verificando se ele existe entre os números que já foram fornecidos. Exibir na tela o vetor final que foi digitado.

---

29. Faça um programa que leia dez conjuntos de dois valores, o primeiro representando o número do aluno e o segundo representando a sua altura em metros. Encontre o aluno mais baixo e o mais alto. Mostre o número do aluno mais baixo e do mais alto, juntamente com suas alturas.

---

30. Faça um programa que leia dois números a e b (positivos menores que 10000) e:Crie um vetor onde cada posição é um algarismo do número. A primeira posição é o algarismo menos significativo;Crie um vetor que seja a soma de a e b, mas faça-o usando apenas os vetores construídos anteriormente. Dica: some as posições correspondentes. Se a soma ultrapassar 10, subtraia 10 do resultado e some 1 à próxima posição.

---

31. Faça um programa que leia dois números n e m e:Crie e leia um vetor de inteiros de n posições;Crie e leia um vetor de inteiros de m posições;Crie e construa um vetor de inteiros que seja a interseção entre os dois vetores anteriores, ou seja, que contém apenas os números que estão em ambos os vetores.Crie e construa um quarto vetor de inteiros que seja a uniãoo entre os dois vetores anteriores lidos, ou seja, que contém os elementos dos dois vetores.

---

32. Faça um programa que receba o nome de n clientes e armazene-os em um vetor. Em um segundo vetor, armazene a quantidade de DVDs locados em 2009 por cada um dos clientes. Sabe-se que, para cada dez locações, o cliente tem direito a uma locação grátis. Faça um programa que mostre o nome de todos os clientes, com a quantidade de locações grátis a que ele tem direito.

---

33. Faça um programa que preencha três vetores com n posições cada um: o primeiro vetor, com os nomes dos produtos; o segundo vetor, com os códigos dos produtos; e o terceiro vetor; com os preços dos produtos. Mostre um relatório apenas com o nome, o código, o preço e o novo preço dos produtos que sofrerão aumento. Sabe-se que os produtos que sofrerão aumento são aqueles que possuem código par ou preço superiora R$ 1.000,00. Sabe-se ainda que, para os produtos que satisfizerem às duas condições anteriores, código e preço, o aumento será de 20%; para aqueles que satisfazerem apenas a condição de código, o aumento será de 15%; e aqueles que satisfazerem apenas a condição de preço, o aumento será de 10%.

---

34. Faça um vetor de tamanho 50 preenchido com o seguinte valor: $(i + 5i)\%i$, sendo i aposição do elemento no vetor, em seguida imprima o vetor na tela.

---

35. Faça um programa que calcule o desvio padrão (STD) de um vetor v contendo n números, ondem m a média do vetor.

$$STD = \sqrt{\frac{1}{n-1}\sum_{i=0}^{n-1}(v[i] - m)^2}$$

6 Processando a Informação: Cap. 6: Matrizes

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

6.1 Sumário

- Revisão do capítulo anterior
- Introdução
- Instanciando matrizes
- Acessando elementos de uma matriz
- Formas de percorrer uma matriz
- Aplicações usando matrizes
- Revisão deste capítulo
- Exercícios

6.2 Revisão do capítulo anterior (Vetores)

- Introdução > Vetores são estruturas para armazenar vários elementos de um mesmo tipo de dados em uma única variável.
- Trabalhando com vetores > Cada linguagem possui uma sintaxe própria para declarar e alocar vetores.
- Acessando elementos de um vetor > **ATENÇÃO** para não acessar uma posição do vetor não reservada/alocada, geralmente `<0` e `>=n`.
- Formas de percorrer um vetor > É possível varrer um vetor na forma ***raster*** e ***anti-raster***, também usando diferentes passos, mas geralmente é passo=1.
- Modularização e vetores > Muito útil usar principalmente os módulos de `leiaVetor` e `escrevaVetor`, podendo ser reaproveitados em vários códigos.

- Tudo que foi visto no capítulo anterior de **Vetores** (ou estruturas **unidimensional** ou **1D**) se estende a estrutura de **Matrizes** (ou estruturas **bidimensional** ou **2D)**).

6.3 Introdução

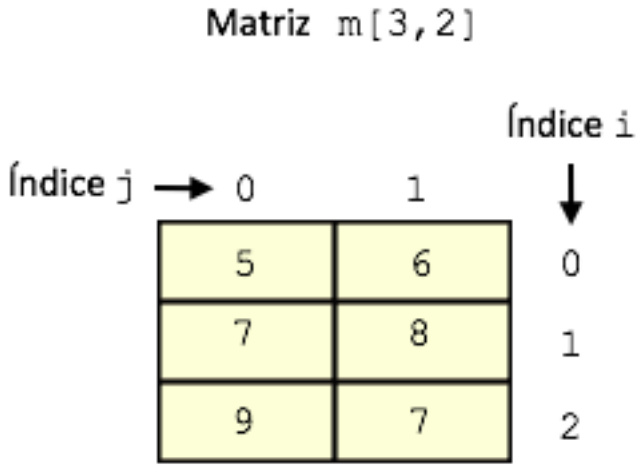
- Existem situações onde é necessário estender a definição de vetor para mais de uma dimensão de dados.
- Por exemplo, uma forma muito utilizada de manipulação de dados é em tabelas.
- Como exemplo, para listar todos os alunos de uma turma e suas notas em uma disciplina,
  - as linhas da tabela podem armazenar a identificação dos alunos na primeira coluna,
  - nas colunas seguintes podem armazenar as notas da prova1, prova2, projeto e, numa última coluna, podem armazenar a nota final do aluno.
- Analogamente a um vetor, em uma matriz cada elemento possui apenas um dado.
- Além disso, na maioria das linguagens de programação, todos os elementos de uma matriz são de um mesmo tipo de dado.
- Um outro exemplo de matrizes muito usado, especialmente com a popularização dos dispositivos móveis como celulares e *tablets*, que comumente trazem câmeras digitais acopladas, ocorre no armazenamento e processamento digital de imagens.
- Mas antes das câmeras digitais, já havia *scanners* e digitalizadores, e uma imagem pode ser representada por uma **matriz** em um computador.
- Veja o conteúdo no ***QRCode*** (imagem=matriz) abaixo usando um aplicativo leitor de QRCode, disponível para *Smartphones* (teste nesta imagem).
- O conteúdo do ***QRCode*** direciona para uma página *web* mostrando uma imagem colorida.
- As imagens coloridas podem ser armazenada em uma estrutura de matriz **tridimensional**, onde cada dimensão armazena uma matriz para uma cor primária (RGB - *Red-Green-Blue* ou vermelhor, verde e azul).

6.4 Instanciando Matrizes

- Nas linguagens MatLab e R, uma matriz é definida com elementos da posição `(1,1)` até a posição `(L, C)`, onde `L` representa o número de linhas e `C` representa o número de colunas de uma matriz.
- Na maioria das linhagens de programação, no entanto, o índice começa no zero, sendo os elementos de uma matriz armazenados da posição `(0,0)` até `(L-1,C-1)`.
- Nos exemplos a seguir são apresentados alguns exemplos de instanciação de matrizes em diferentes linguagens de programação.

6.5 Acessando elementos de uma matriz

- Uma matriz está pronta para se inserir elementos ou se alterar seus dados após instanciada e alocada na memória, em todas as suas dimensões.
- Veja uma forma de visualizar uma matriz 2D na Figura abaixo.



- Essa figura ilustra uma matriz com `3` linhas e `2` colunas.
- Para visualizar seus elementos podemos usar os índices `i` e `j`,
  - com valores para as linhas `0<=i<3` e
  - para as colunas `0<=j<2`.
- Analogamente ao vetor, mas com uma dimensão a mais, a matriz recebe valores para os seus elementos, conforme a seguinte estrutura em pseudocódigo:

```
Instanciar uma matriz m com 3 linhas e 2 colunas
m[0,0] = 5 # linha i=0
m[0,1] = 6
m[1,0] = 7 # linha i=1
m[1,1] = 8
m[2,0] = 9 # linha i=2
m[2,1] = 7
```

6.6 Formas de se percorrer uma matriz

- Analogamente ao vetor, uma matriz, após criada, possui tamanho fixo.
- Geralmente, para cada dimensão da matriz, é recomendado usar uma estrutura de repetição `para`, > com o objetivo de tornar o código mais compacto e genérico para matrizes de quaisquer dimensões.
- Além disso, é natural percorrer (ou varrer) a matriz
  - da linha `i=0` até a linha `i<L`, onde L representa o número de linhas de uma matriz.
  - e da coluna `j=0` até a coluna `j<C`.
- No exemplo a seguir em pseudocódigo, uma matriz `m` é criada com `6` elementos, sendo três linhas e `2` colunas, e os dois laços `para` inicializam todos os seus elementos com o valor 0.

```
Instanciar uma matriz m com 3 linhas e 2 colunas
inteiros L=3, C=2
Para cada i, de i=0; até i<L; passo i=i+1 faça
    Para cada j, de j=0; até j<C; passo j=j+1 faça
        m[i,j] = 0
```

- Existem várias formas de percorrer (ou varrer) uma matriz usando uma estrutura de repetição.
- Por exemplo, é possível percorrer uma matriz
  - do primeiro elemento `m[0,0]` > (convencionando como canto superior esquerdo)
  - até o último elemento `m[L-1,C-1]` > (convencionando como canto inferior direito da matriz),
  - linha por linha, ou coluna por coluna.
- Esse tipo de varredura é chamada ***raster***.
- A varredura inversa é chamada ***anti-raster***, do último elemento inferior direito, até o primeiro elemento superior esquerdo.

6.7 Exemplo 01 - Ler/Escrever matriz

- Analogamente ao que foi feito no capítulo anterior sobre vetores, onde alocamos os vetores em tempo de execução através de métodos, é possível usar modularização para melhorar a organização, manutenção e reaproveitamento de código.
- Aqui é apresentado um método `leiaMatriz` e `escrevaMatriz` genéricos
- Para entrada de dados, ou seja, inserir valores nos elementos alocados na memória para uma matriz.
- Além de saída de dados, para escrever a matriz, linha por linha.

Pseudocódigo Exemplo 01: Considere um algoritmo para: * Ler um inteiro `L` (linhas) representando o número de alunos, * Ler um inteiro `C` representando o número de avaliações. * Considere a primeira coluna o `RA` do aluno, assim `C=C+1`. * Criar uma matriz `m` com dimensões `LxC`. * Ler todos os elementos da matriz. * Escrever todos os

elementos da matriz, formando a saída, linha por linha, por exemplo, para uma matriz com 2 alunos e 3 avaliações, escreva:

```
LISTA DE ALUNOS vs Avaliações:
1234 4 3 9
3456 6 4 8
```

```
Função inteiro m[][] leiaMatriz(inteiro L, inteiro C):
    Instanciar e alocar uma matriz m de Reais com L x C
    Para cada i, de i=0; até i<L; passo i=i+1 faça
        Para cada j, de j=0; até j<C; passo j=j+1 faça
            m[i,j] = leia("Digite um número inteiro:");

Função escrevaMatriz(inteiro m[][], inteiro L, inteiro C):
    Instanciar e alocar uma matriz m de Reais com L x C
    Para cada i, de i=0; até i<L; passo i=i+1 faça
        Para cada j, de j=0; até j<C; passo j=j+1 faça
            escreva(" ", m[i,j]);
        escreva("\n"); // pula linha

// PROGRAMA PRINCIPAL
// ENTRADAS
inteiro L = leia("Digite o numero de alunos:")
inteiro C = leia("Digite o numero de avaliações:")
C = C + 1  // a primeira coluna é o RA
Instanciar uma matriz m com L linhas e C colunas

m = leiaMatriz(L,C)
// PROCESSAMENTO: ?

// SAÍDA
escrevaMatriz(m)
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=2
3
1234
4
3
9
3456
6
4
8
output=
```

```
LISTA DE ALUNOS vs Avaliações:
1234 4 3 9
3456 6 4 8
```

[ ]:
```
%%writefile cap6ex01.c
#include <stdio.h>
#include<malloc.h>

int ** leiaMatriz(int L, int C) {
  int **m = (int **)malloc(L*sizeof(int*));
  for (int i = 0; i < L; i++) {
    m[i] = (int *)malloc(C * sizeof(int)); // for each row allocate C␣
 ↪ints
    for (int j = 0; j < C; j++)
      scanf("%d", &m[i][j]);
  }
  return m;
}
void free_matrix(int **m, int L) {
    for (int i = 0; i < L; i++)
         free(m[i]);
    free(m);
 }
void escrevaMatriz(int **m, int L, int C) {
  for (int i = 0; i < L; i++) {
   for (int j = 0; j < C; j++)
      printf("%d\t", m[i][j]);
   printf("\n");
  }
}
int main(void) {
  // ENTRADA DE DADOS
  int L, C, **m;   // variaveis de referência m
  printf("Digite o número de alunos: ");
  scanf("%d", &L);

  printf("Digite o número de avaliações: ");
  scanf("%d", &C);

  C = C + 1; // a primeira coluna é o RA do aluno

  printf("Digite os elementos da matriz");
  m = leiaMatriz(L,C);

  // PROCESSAMENTO ?

  // SAÍDA DE DADOS
  printf("\nLISTA DE ALUNOS vs Avaliações:\n");
```

```
  printf("RA ");
  for (int i = 0; i < C-1; i++)
    printf("\t%d",(i+1));

  printf("\n");
  escrevaMatriz(m,L,C);
  free_matrix(m,L); // liberar memória alocado com malloc
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap6ex01.c -o output2
./output2
```

Acessando elemento com `*` Outras forma de acessar elementos em matriz.

```
for (int i = 0; i < L; i++)
 for (int j = 0; j < C; j++) {
    printf("%d\t", m[i][j]); // ou
    printf("%d\t", *(m + i*C + j));
 }
```

Ou utilizando apenas um laço para varrar uma matriz:

```
for (int i = 0; i < L*C; i++) {
    printf("%d\t", m[i/C][i%C]); // ou
    printf("%d\t", *(m + i));
 }
```

Matriz Multimensional

```
for (int k = 0; k < D; k++) // profundidade
  for (int i = 0; i < L; i++) // linha
   for (int j = 0; j < C; j++) { // coluna
      printf("%d\t", m[k][i][j]); // ou
      printf("%d\t", *(m + k*L*C + i*C + j));
   }
```

Ou utilizando apenas um laço para varrar uma matriz:

```
for (int i = 0; i < D*L*C; i++) {
    d = i/(L*C);
    printf("%d\t", m[d][(d-i)/C][(d-i)%C]); // ou
    printf("%d\t", *(m + i));
 }
```

6.8 Exercícios

Ver notebook Colab nos arquivos `cap6.partX.lab.*.ipynb` (`X` ∈ `[2,3,4,5]` e `*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência,

onganizadas em subpastas contidas em `"gen"`, na pasta do Google Drive colabs.

6.9 Atividades no Moodle+VPL

Algumas atividades no Moodle+VPL pedem como entradas matrizes de inteiros (ou reais), **armazenados em várias linhas**. Exemplo de entrada a ser lida:

6.9.1 Entrada de Dados (cada linha contem um texto ou *string* com elementos da linha da matriz e vários espaços “”):

```
0 9 3 6 9 8 4 5 4
8 1 2 3 5 2 9 9 6
4 1 1 0 9 9 8 2 7
5 2 8 4 6 6 0 8 0
4 7 6 4 3 9 3 3 5
2 6 0 4 0 7 5 5 2
9 8 4 8 4 7 1 4 3
```

Para não ter que incluir várias entradas inteiras, a melhor solução é fazer um método de leitura, passando como argumento um texto (*string*) com várias linhas. O final de cada linha é definido por `\n` e o final da matriz deve ser uma linha em branco com apenas `\n`. Esse método deve retornar a matriz.

6.10 Revisão deste capítulo de Matriz

- Introdução
- Instanciando matrizes
- Acessando elementos de uma matriz
- Formas de percorrer uma matriz
- Aplicações usando matrizes
- Exercícios
- Revisão deste capítulo de Matriz

6.11 Processando a Informação: Cap. 6: Matrizes - Prática 1

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

6.11.1 Exercícios

---

1. Criar uma matriz para armazenar as `C` notas de uma turma e calcular a média de cada aluno, considerando uma turma com `L` alunos.

---

2. Em complemento ao exercício anterior, criar também um vetor de String, onde para cada posição `0<=i<L` seja armazenada a palavra `"reprovado"` se a nota for abaixo de 5, ou `"aprovado"`, caso contrário.

---

3. Criar uma matriz para armazenar `L` linhas e `C` colunas. Calcular e exibir a soma dos elementos da diagonal principal e secundária da matriz.

---

4. Criar uma matriz para armazenar `L` linhas e `C` colunas. Calcular a soma dos elementos acima da diagonal da matriz.

---

5. Criar uma matriz `m1` para armazenar `L` linhas e `C` colunas. Criar uma outra matriz `m2` para armazenar `m1` transposta, com `C` linhas e `L` colunas, sendo que a primeira linha de `m2` contenha os elementos da primeira coluna de `m1`. Isso se repete para as demais linhas de `m2` e colunas de `m1`.

6.12 Processando a Informação: Cap. 6: Matrizes - Prática 2

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

6.12.1 Exercícios

---

1. Criar uma função/método para calcular e retornar o determinante de uma matriz `3 x 3` de valores reais recebida como parâmetro.

---

2. Criar um método que receba uma matriz qualquer de inteiros e imprima na tela o valor máximo e o valor mínimo encontrados (não retornando nada).

---

3. Criar um método que receba uma matriz de inteiros de qualquer dimensão e retorne um vetor contendo a somatória dos valores contidos nas colunas.

4.Considere:

- a) Criar um método que receba uma matriz `m1` de inteiros positivos com `L` linhas e `C` colunas e retorne uma matriz `m2`, de mesma dimensão, onde em cada posição `[i,j]` seja atribuído o cálculo dos máximos entre o elemento de `m1` e seus oito vizinhos:

vizinhança		
`[i-1,j-1]`	`[i-1,j]`	`[i-1,j+1]`
`[i ,j-1]`	`[i ,j]`	`[i ,j+1]`
`[i+1,j-1]`	`[i+1,j]`	`[i+1,j+1]`

- b) Generalize este código para os `m` vizinhos da `m1` à esquerde e à direita e os n vizinhos acima e abaixo.

---

5. Criar um método que receba duas matrizes de valores reais e retorne à multiplicação das duas matrizes.

6.13 Processando a Informação: Cap. 6: Matrizes - Prática 3

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

6.13.1 Exercícios

Fonte: ref1; ref2

---

1. Uma matriz quadrada NxN lógica representa as posições minadas de um jogo. Quando uma posição possui o valor VERDADEIRO significa que há uma mina ali. Escreva um programa que informa se é possível percorrer o tabuleiro de um lado ao outro em linha reta (atravessando uma linha inteira ou coluna inteira) ou diagonal sem passar por uma mina sequer.

---

2. Uma imagem em níveis de cinza pode ser representada por uma matriz. Leia uma imagem A de números inteiros. Leia também um limiar inteiro. Fazer um método para calcular e retornar uma imagem binária de saída B com as mesmas dimensões de A, considerando para cada pixel A(i,j)>limear, B(i,j) deve receber o valor 1, caso contrário recebe o valor 0. Imprimir as imagens de entrada e saída.

---

3. Leia uma matriz A de inteiros. Criar uma matriz B considerando que cada elemento de B é o resultado do elemento de A multiplicado pela média dos elementos da linha deste elemento. Arredondar cada elemento com o comando `round`.

---

4. Faça um programa para gerar automaticamente números entre 0 e 99 de uma cartela de bingo. Sabendo que cada cartela deverá conter 5 linhas de 5 números, gere estes

dados de modo a não ter números repetidos dentro das cartelas. O programa deve exibir na tela a cartela gerada.

---

5. Leia uma matriz 5 x 10 que se refere respostas de 10 questõs de múltipla escolha, referentes a 50 candidatos de um processo seletivo. Leia também um vetor de 10 posições contendo o gabarito de respostas que podem ser a, b, c, d ou e. Seu programa deverá comparar as respostas de cada candidato com o gabarito e emitir um vetor denominado resultado, contendo a pontuação correspondente a cada candidato. Imprimir também a lista de aprovados e em seguinda a lista de reprovados, considernado média 5.

---

6. Leia uma matriz A de inteiros. Qual é o maior produto de quatro números adjacentes em qualquer direção (cima, baixo, esquerda, direita, ou nas diagonais) na matriz de 20x20 e em qual(is) coordenada(s) ocorreu(ram)? Ou seja, em cada posição `[i,j]` de A cálcular o máximo valor entre as multiplicações dos elementos abaixo:

vizinhança		
`[i-1,j-1]`	`[i-1,j]`	`[i-1,j+1]`
`[i ,j-1]`	`[i ,j]`	`[i ,j+1]`
`[i+1,j-1]`	`[i+1,j]`	`[i+1,j+1]`

---

7. Uma matriz de caracteres 3x3 foi utilizada para armazenar uma partida de jogo da velha. Os caracteres ‘O’ e ‘X’ foram utilizados para armazenar a jogada de cada participante. Informe na tela se o vencedor foi o jogador ‘O’, o jogador ‘X’ ou se o resultado foi empate. IMPORTANTE: não serão informadas partidas com dois vencedores, apenas partidas válidas e todas as 9 casas estarão preenchidas com ‘O’ ou ‘X’.

---

8. Faça um programa para determinar a próxima jogada em um Jogo da Velha. Assumir que o tabuleiro é representado por uma matriz de 3 x 3, onde cada posição representa uma das casas do tabuleiro. A matriz pode conter os seguintes valores -1, 0, 1 representando respectivamente uma casa contendo uma peça minha (-1), uma casa vazia do tabuleiro (0), e uma casa contendo uma peça do meu oponente (1).

6.14 Processando a Informação: Cap. 6: Matrizes - Prática 4

Este caderno (Notebook) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.

6.14.1 Exercícios

EP5_1 - Média dos alunos

Ler uma matriz LINHA por LINHA:

Ler uma matriz considerando cada ELEMENTO da matriz uma entrada:

7 Processando a Informação: Cap. 7: Tipos Definidos Pelo Programador e Arquivos

Este caderno (Notebook em CONSTRUÇÃO) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.
>
> O conteúdo deste capítulo foi inspirado em: * Cap. 7: Conceitos de programação orientada a objetos do livro anterior * Notas de aula de professores da UFABC, em especial, dos professores Luiz Rozante e Wagner Botelho.

7.1 Sumário

- Revisão do capítulo anterior
- Introdução
- Paradigma Estruturado
- Paradigma Orientado a Objetos
- Tipos de dados
- Arquivos
- Revisão deste capítulo
- Exercícios

7.2 Revisão do capítulo anterior (Matriz)

- Introdução
- Instanciando matrizes
- Acessando elementos de uma matriz
- Formas de percorrer uma matriz
- Aplicações usando matrizes
- Exercícios

7.3 Introdução

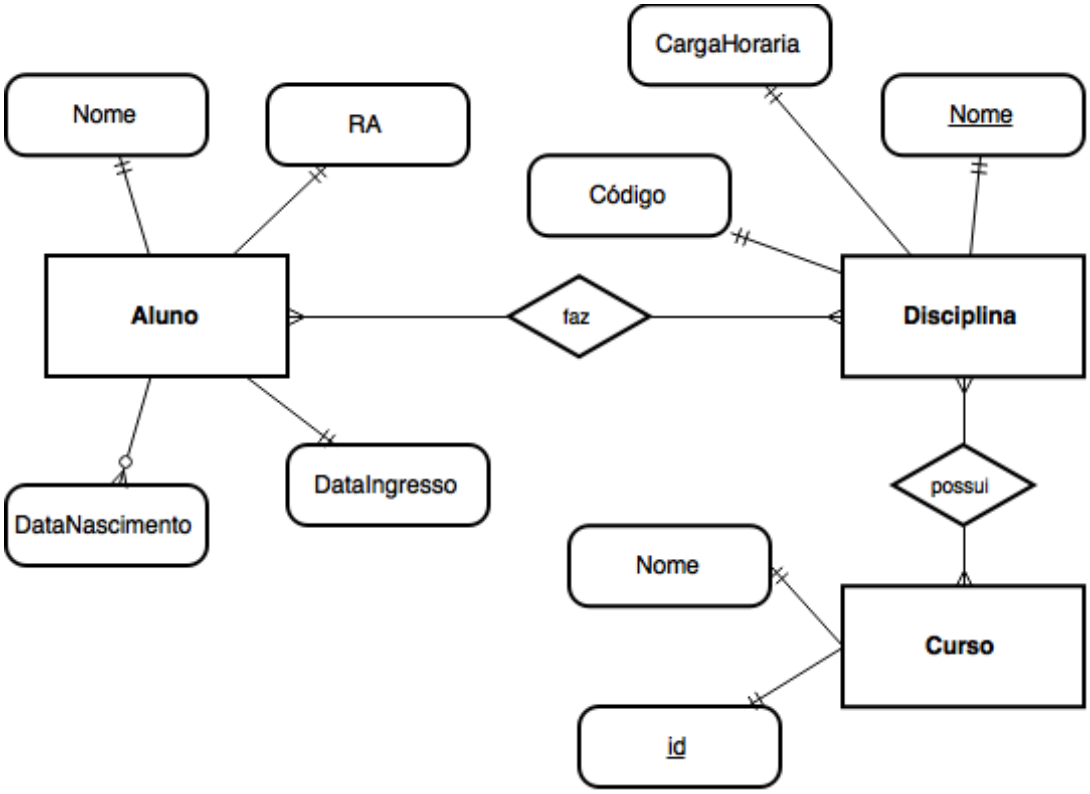
- Este capítulo introduz alguns conceitos usados em:
  - Programação Estruturada,
  - Programação Orientada a Objetos (POO) e
  - Engenharia de Software (ES).
- Esses conceitos serão úteis em estudos futuros.
- Não é o propósito deste capítulo cobrir completamente estes temas, mas sim proporcionar um ponto de partida para o estudo avançados em programação.
- Além de introduzir processos para desenvolver códigos envolvendo equipes e utilizando princípios da Engenharia de Software.

7.4 Paradigma Estruturado

- Uns dos primeiros estilos (ou paradigma) para estruturar um Software é chamado de **Programação Estruturada**.
- Neste paradigma estruturado, o programador define **Tabelas** (ou **Registros** ou também chamadas **Entidades**), onde se podem armazenar variáveis de vários tipos de dados,
  - diferentemente dos vetores e matrizes vistos nos capítulos anterios, onde todos os dados devem ser de um mesmo tipo de dado (exceto as listas em Python).
- Por exemplo, é possível definir uma tabela para armazenar as informações de um aluno num contexto de um sistema acadêmico, chamada **Aluno**.
- Esta tabela **Aluno**, na verdade, pode ser considera um novo tipo de dados e é possível, por exemplo, "instanciar" uma variável (ou registro), como *Julia* do tipo **Aluno**.
- Como atributos da tabela **Aluno**, é possível ter *nome*, *matrícula*, *data de nascimento*, *ano de ingresso*, etc.
- Observe que na tabela **Aluno** é importante também ter informação de curso e de disciplinas já cursadas.
- Estas informações podem ser "instâncias" de outras tabelas, como **Curso** e **Disciplina**, com seus respectivos atributos apropriados.
- Em Banco de Dados estas "instâncias" são possíveis através de um atributo representando a chave (estrangeira) de outra tabela.
- Além disso, cada tabela possui um atributo (chamado chave primária) que identifica o registro, por exemplo, Julia, que pode estar armazenada num registro de número 33.
- Resumindo, as **estrutura** definidas pelo programador podem armazenar em tempo de execução as **tabelas** definidas em Bancos de Dados (com informações armazenadas em disco).

7.5 Diagrama Entidade Relacionamento

- Para poder visualizar melhor esta relação entre as tabelas **Aluno**, **Curso** e **Disciplina**, existe um modelo apropriado, chamado **Diagrama Entidade Relacionamento (DER)**, como apresentado na figura abaixo.
- Observe o relacionamento "**Aluno** faz **Disciplina**(s)".
- Se for considerar um contexto onde um aluno está cursando uma disciplina, existe a necessidade de incluir uma nova tabela, chamada, por exemplo, **Turma**.
- Assim, "**Turma** possui **Aluno**(s)" e "**Disciplina** possui Turma(s)".

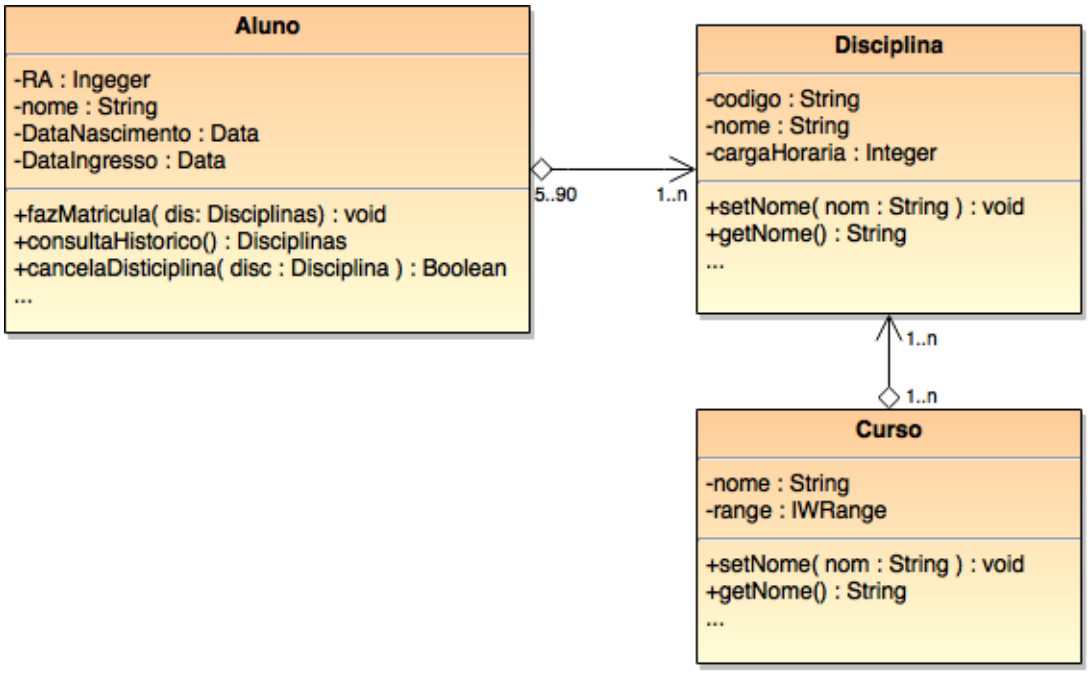


- Desse modo, para um sistema acadêmico completo, é necessário criar várias tabelas e seus relacionamentos.
- Todos esses dados das tabelas devem ser lidos (digitados) uma vez e armazenados em disco.
- Senão, toda vez que esse sistema acadêmico for executado, todos os dados deverão ser lidos novamente.
- Existem várias formas de guardar os dados de variáveis em disco:
  - arquivo texto,
  - arquivo binário ou
  - tabelas em um Sistema de Gerenciamento de Banco de Dados (SGBD), como Oracle, JDBC, SQL, PostgreSQL, etc.

- Nas disciplinas de Banco de Dados, o aluno aprende como criar essas tabelas em um SGBD e também como usá-las dentro de uma linguagem de programação.
- Essa forma de organizar vários tipos de dados em tabelas chama-se **encapsulamento de dados**.
- O grande desafio dos programadores e analistas de sistemas é definir corretamente essas tabelas e os seus relacionamentos. Os sistemas bem modelados são mais fáceis de realizar manutenções.
- Além disso, os programadores devem criar estruturas, funções e procedimentos de forma bem organizada, por exemplo, usando vários arquivos organizados em pastas.
- O paradigma estruturado não facilita completamente esta organização. Isso já não ocorre no paradigma orientado a objetos, resumido a seguir.

7.6 Paradigma Orientado a Objetos

- Na programação estruturada não existe uma forma eficiente de organizar (encapsular) as funções específicas de uma tabela, como ocorre no encapsulamento dos dados, visto na programação estruturada.
- Assim, surgiu a necessidade de criar um novo paradigma de programação, que é a **Programação Orientada a Objetos (POO)**, onde, além de encapsular os dados, é possível encapsular as funções ou métodos que os processam.
- Na POO, tabela passou a se chamar **Classe**, e um registro de uma tabela passou a ser chamado de **instância** de uma classe, também chamado de **objeto**.
- Como no exemplo anterior, para o sistema acadêmico, é possível criar a classe **Aluno** (que é um novo tipo de dados) e com ela instanciar uma variável *Julia* da classe **Aluno**.



O restante deste capítulo é complemetar ao livro **Processando a Informação: um livro prático de programação independente de linguagem**, apresentando conceitos e exemplos de Programação Estruturada.

7.7 Tipos de dados

- Os tipos de dados em C são:
  - `struct`

```
struct MEU_TIPO{
 tipo1 nome1;
 tipo2 nome2;
 ...
};
struct MEU_TIPO Entidade1, Entidade2;
```

  - Ou melhor, com `typedef` (utilizado para renomear um tipo de dado da própria linguagem ou definido pelo programador):

```
typedef struct {
tipo1 nome1;
tipo2 nome2;
...
} MEU_TIPO;
MEU_TIPO Entidade1, Entidade2;
```

  - `union`
  - `enum`

7.7.1 Exemplo 01 - Criar um registro de Aluno

Exemplo para criar a `struct` **TAluno** contendo 4 atributos.

[ ]:
```
%%writefile cap7ex01.c
#include <stdio.h>
#include <string.h>

struct TAluno {
  char nome[50];
  int idade;
  char rua[50];
  int numero;
};

int main() {
  struct TAluno ana; // instancia uma variável c do tipo TAluno
  // ENTRADA DE DADOS
  strcpy(ana.nome, "Ana Silva");
  ana.idade = 18;
  strcpy(ana.rua, "Avenida Paulista");
```

```
  ana.numero = 1000;

  // SAÍDA DE DADOS
  printf("nome: %s\nidade: %d\n", ana.nome, ana.idade);
  printf("rua: %s\nnúmero: %d\n", ana.rua, ana.numero);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex01.c -o output
./output
```

7.7.2 Exemplo 02 - Criar um registro de Aluno, com `scanf`

Exemplo para criar a `struct` **TAluno** contendo 4 atributos lidos do teclado com `scanf`.

[ ]:
```
%%writefile cap7ex02.c
#include <stdio.h>
#include <string.h>

struct TAluno {
  char nome[50];
  int idade;
  char rua[50];
  int numero;
};

int main() {
  struct TAluno Alunos[2]; // instancia uma vetor do tipo TAluno
  // ENTRADA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("Entre com os dados: nome, idade, rua, número:\n");
    fflush(stdin);
    fgets(Alunos[i].nome, 50, stdin);
    scanf("%d", &Alunos[i].idade);
    fflush(stdin);
    fgets(Alunos[i].rua, 50, stdin);
    scanf("%d", &Alunos[i].numero);
  }
  // SAÍDA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("nome: %s\nidade: %d\n", Alunos[i].nome, Alunos[i].idade);
    printf("rua: %s\nnúmero: %d\n", Alunos[i].rua, Alunos[i].numero);
  }
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex02.c -o output
./output
```

7.7.3 Exemplo 03 - Criar um registro de Aluno, com `typedef`

Exemplo para criar a `struct` **TAluno** contendo 4 atributos lidos do teclado, renomeando com `typedef` para **Aluno**.

[ ]:
```
%%writefile cap7ex03.c
#include <stdio.h>
#include <string.h>

struct TAluno {
  char nome[50];
  int idade;
  char rua[50];
  int numero;
};

typedef struct TAluno Aluno;

int main() {
  // ENTRADA DE DADOS
  Aluno Ana = { "Ana Silva" , 18, "Avenida Paulista" , 1000 };

  // SAÍDA DE DADOS
  printf("%s %d %s %d", Ana.nome, Ana.idade, Ana.rua, Ana.numero);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex03.c -o output
./output
```

7.7.4 Exemplo 04 - Criar um registro de Aluno, com `typedef`, opção 2

Exemplo para criar a `struct` **Aluno** de forma simplificada, contendo 4 atributos.

[ ]:
```
%%writefile cap7ex04.c
#include <stdio.h>
#include <string.h>

typedef struct TAluno {
  char nome[50];
  int idade;
  char rua[50];
  int numero;
```

```
} Aluno;

int main() {
  // ENTRADA DE DADOS
  Aluno ana = { "Ana Silva" , 18, "Avenida Paulista" , 1000 };

  // SAÍDA DE DADOS
  printf("%s %d %s %d", ana.nome, ana.idade, ana.rua, ana.numero);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex04.c -o output
./output
```

7.7.5 Exemplo 05 - Criar dois registros

Exemplo para criar a `struct` **Cliente** contendo 3 atributos, sendo um deles uma outra estrutura **Endereco**.

[ ]:
```
%%writefile cap7ex05.c
#include <stdio.h>
#include <string.h>

typedef struct {
  char rua[50];
  int numero;
} Endereco;

typedef struct {
  char nome[50];
  int idade;
  Endereco end;
} Cliente;

void leiaCliente(Cliente* cliente) {
  fflush(stdin);
  printf("nome: ");
  fgets((*cliente).nome, 50, stdin);
  printf("idade: ");
  scanf("%d", &(*cliente).idade);
  fflush(stdin);
  printf("rua: ");
  fgets((*cliente).end.rua, 50, stdin);
  printf("numero: ");
  scanf("%d", &(*cliente).end.numero);
}
```

```
void escrevaCliente(Cliente cliente) {
  printf("nome: %s\nidade: %d\n", cliente.nome, cliente.idade);
  printf("rua: %s\nnumero: %d\n", cliente.end.rua, cliente.end.numero);
}

int main() {
  Cliente clientes[2];
  // ENTRADA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("Cadastrar dados do cliente %d\n", i + 1);
    leiaCliente(&clientes[i]);
  }

  // SAÍDA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("Imprimir dados do cliente %d\n", i + 1);
    escrevaCliente(clientes[i]);
  }
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex05.c -o output
./output
```

7.7.6 Exemplo 06 - Criar dois registros usando biblioteca

Exemplo para criar a `struct` **Cliente** contendo 3 atributos, sendo um deles uma outra estrutura **Endereco**, organizados em uma biblioteca.

[ ]:
```
%%writefile cap7ex06.c
#include "myBiblioteca.h"

int main() {
  Cliente clientes[2];
  // ENTRADA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("Cadastrar dados do cliente %d\n", i + 1);
    leiaCliente(&clientes[i]);
  }

  // SAÍDA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("Imprimir dados do cliente %d\n", i + 1);
    escrevaCliente(clientes[i]);
  }
  return 0;
}
```

[ ]:
```
%%writefile myBiblioteca.h
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct {
  char rua[50];
  int numero;
} Endereco;

typedef struct {
  char nome[50];
  int idade;
  Endereco end;
} Cliente;

void leiaCliente(Cliente* cliente);
void escrevaCliente(Cliente cliente);
```

[ ]:
```
%%writefile myBiblioteca.c
#include "myBiblioteca.h"

void escrevaCliente(Cliente cliente) {
  printf("nome: %s\nidade: %d\n", cliente.nome, cliente.idade);
  printf("rua: %s\nnúmero: %d\n", cliente.end.rua, cliente.end.numero);
}

// leia um cliente a partir do teclado
// nome, idade, rua, numero
void leiaCliente(Cliente* cliente) {
  fflush(stdin);
  printf("nome: ");
  fgets((*cliente).nome, 50, stdin);
  printf("idade: ");
  scanf("%d", &(*cliente).idade);
  fflush(stdin);
  printf("rua: ");
  fgets((*cliente).end.rua, 50, stdin);
  printf("numero: ");
  scanf("%d", &(*cliente).end.numero);
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex06.c myBiblioteca.c -o output
./output
```

7.8 Arquivos

Os arquivos armazenados em disco são considerados como uma sequência de caracteres e podem ser acessados em memória através de um ponteiro para o primeiro caracter do arquivo (texto ou binário), como segue:

```
FILE *fp;
fp = fopen("nomeArquivo.txt", "modo"); // modo é definido a seguir
```

Seguem as principais funções para manipular arquivos da biblioteca `stdio.h`:

Função	Descrição
`fopen()`	Abre arquivo
`fclose()`	Fecha arquivo
`putc()`	Escreve um caracter no arquivo
`fputc()`	Igual `putc()`
`getc()`	Lê um caracter do arquivo
`fgetc()`	Igual `getc()`
`fseek()`	Posiciona o arquivo em um determinado byte
`fprintf()`	Semelhante ao `printf()`, mas para arquivo
`fscanf()`	Semelhante ao `scanf()`, mas para arquivo
`feof()`	Verifica final de arquivo
`ferror()`	Verifica se ocorreu erro
`rewind()`	Posiciona o ponteiro para o início do arquivo
`remove()`	Apaga arquivo
`fflush()`	Descarrega o *buffer* do arquivo
`fgets()`	Obtém uma string do arquivo
`fread()`	Lê um bloco de dados do arquivo
`fwrite()`	Escreve um bloco de dados no arquivo
`ftell()`	Retorna a posição do ponteiro

Modo	**Arquivo**	**Função**
r	Texto	Leitura
w	Texto	Escrita em arquivo novo
a	Texto	Escrita em arquivo existente
r+	Texto	Leitura/Escrita
w+	Texto	Leitura/Escrita
a+	Texto	Leitura/Escrita
rb	Binário	Leitura
wb	Binário	Escrita
ab	Binário	Escrita
r+b	Binário	Leitura/Escrita
w+b	Binário	Leitura/Escrita
a+b	Binário	Leitura/Escrita

7.8.1 Exemplo 07 - Criar Arquivo

Exemplo para criar um arquivo texto.

[ ]:
```
%%writefile cap7ex07.c
#include <stdio.h>

int main() {
  char* filename = "teste.txt";
  FILE* file;

  file = fopen(filename, "w"); // Cria arquivo para escrita
  if (file == NULL) {
    printf("Erro ao abrir o arquivo: %s\n", filename);
    return -1;
  }

  // SAÍDA DE DADOS
  for (int i = 0; i < 10; i++)
    fprintf(file, "Linha %02d\n", i + 1); // Escrever algo no arquivo

  fclose(file); // fecha arquivo
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex07.c -o output
./output
cat teste.txt
```

7.8.2 Exemplo 08 - Ler arquivo

Exemplo para ler um arquivo texto.

[ ]:
```
%%writefile cap7ex08.c
#include <stdio.h>

int main() {
  char* filename = "teste.txt", ch;
  FILE* file;

  // ENTRADA DE DADOS
  file = fopen(filename, "r"); // Cria arquivo para leitura
  if (file == NULL) {
    printf("Erro ao abrir o arquivo: %s\n", filename);
    return -1;
  }

  // SAÍDA DE DADOS
  while ((ch = fgetc(file)) != EOF) { // Lê arquivo
    printf("%c", ch); // caracter por caracter
  }
```

```
  fclose(file); // fecha arquivo
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex08.c -o output
./output
```

7.8.3 Exemplo 09 - Criar dois registros usando biblioteca, lendo arquivo

Exemplo para criar a `struct` **Cliente** contendo 3 atributos, sendo um deles uma outra estrutura **Endereco**, organizados em uma biblioteca.

Destacando que os dados serão lidos do arquivo CSV abaixo:

[ ]:
```
%%writefile dados.csv
Maria Souza, 19, Avenida Paulista, 1000
Pedro Silva, 18, Avenida Rebouças, 2500
```

[ ]:
```
%%writefile cap7ex09.c
#include "myBiblioteca.h"

int main() {
  Cliente clientes[20];
  char* filename = "dados.csv";

  // ENTRADA DE DADOS
  FILE* file; // Cria arquivo para leitura
  if ((file = fopen(filename, "r")) == NULL) {
    printf("Erro ao abrir o arquivo: %s\n", filename);
    return -1;
  }

  int contador = 0;
  char linha[512]; // espaço para cada linha lida
  while (fgets(linha, sizeof(linha), file))  // para cada linha
    criaCliente(&clientes[contador++], linha);

  // SAÍDA DE DADOS
  for (int i = 0; i < contador; i++) {
    printf("\nImprimir dados do cliente %d\n", i + 1);
    escrevaCliente(clientes[i]);
  }

  fclose(file); // fecha arquivo
  return 0;
}
```

[ ]:
```
%%writefile myBiblioteca.h
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct {
  char rua[50];
  int numero;
} Endereco;

typedef struct {
  char nome[50];
  int idade;
  Endereco end;
} Cliente;

void leiaCliente(Cliente* cliente);
void escrevaCliente(Cliente cliente);
void criaCliente(Cliente* cliente, char linha[512]);
```

[ ]:
```
%%writefile myBiblioteca.c
#include "myBiblioteca.h"
// cria um cliente a partir de uma linha de texto. Exemplo:
// "Maria Souza, 20, Avenida Paulista, 1200"
void criaCliente(Cliente* cliente, char linha[512]) {
  // Ler nome
  char* token = strtok(linha, ","); // ler até a primeira ","
  strcpy((*cliente).nome, token); // nome

  // Ler idade
  token = strtok(NULL, ","); // continar lendo até a próxima ","
  (*cliente).idade = atoi(token); // converter str to int

  // Ler rua
  token = strtok(NULL, ",");
  strcpy((*cliente).end.rua, token + 1); // +1 retira 1o espaço em␣
 ↪branco

  // Ler numero
  token = strtok(NULL, ",");
  (*cliente).end.numero = atoi(token); // converter str to int
}
void escrevaCliente(Cliente cliente) {
  printf("nome: %s\nidade: %d\n", cliente.nome, cliente.idade);
  printf("rua: %s\nnúmero: %d\n", cliente.end.rua, cliente.end.numero);
}
```

```
// leia um cliente a partir do teclado
// nome, idade, rua, numero
void leiaCliente(Cliente* cliente) {
  fflush(stdin);
  printf("nome: ");
  fgets((*cliente).nome, 50, stdin);
  printf("idade: ");
  scanf("%d", &(*cliente).idade);
  fflush(stdin);
  printf("rua: ");
  fgets((*cliente).end.rua, 50, stdin);
  printf("numero: ");
  scanf("%d", &(*cliente).end.numero);
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap7ex09.c myBiblioteca.c -o output
./output
```

7.9 Exercícios

Ver notebook Colab nos arquivos `cap7.partX.lab.*.ipynb` (`X` ∈ `[2,3,4,5]` e `*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, onganizadas em subpastas contidas em `"gen"`, na pasta do Google Drive colabs.

7.10 Revisão deste capítulo

- Introdução
- Paradigma Estruturado
- Paradigma Orientado a Objetos
- Tipos de dados
- Arquivos
- Exercícios
- Revisão deste capítulo de Vetores

8 Processando a Informação: Cap. 8: Ponteiros

Este caderno (Notebook em CONSTRUÇÃO) é parte complementar *online* do livro **Processando a Informação: um livro prático de programação independente de linguagem**, que deve ser consultado no caso de dúvidas sobre os temas apresentados.

> Este conteúdo pode ser copiado e alterado livremente e foi inspirado nesse livro.
>
> O conteúdo deste capítulo foi inspirado em: * Notas do prof. Paulo Feofiloff: [1]; [2]; [3]; [4]. * Notas de aula de professores da UFABC, em especial, dos professores Luiz Rozante e Wagner Botelho. * Para mais detalhes de tipos de alocação de memória, ver [ref].

8.1 Sumário

- Revisão do capítulo anterior
- Introdução
- Alocação estática
- Alocação dinâmica
- Alocação dinâmica para *array* multidimensional
- Tipo Abstrato de Dados (TAD) Lista
- Tipo Abstrato de Dados (TAD) Fila
- Tipo Abstrato de Dados (TAD) Pilha
- Revisão deste capítulo
- Exercícios

8.2 Revisão do capítulo anterior (Struct)

- Introdução
- Paradigma Estruturado
- Paradigma Orientado a Objetos
- Tipos de dados
- Arquivos

- Revisão deste capítulo
- Exercícios

8.3 Introdução

- *Ponteiros* são variáveis especiais que recebem valores referentes à *endereços* da memória principal do computador (RAM - *Random Access Memory*).
- Ao ligar um computador, o sistema operacional é carregado da memória secundária para a RAM.
- Quando criamos um programa em alguma linguagem de programação e executamos, esse programa também é carregado na RAM.
- Uma variável `x=10` criada nesse programa também será enderaçada na RAM, por exemplo, no endereço `FF10AF` (em hexadecimal).
- Algumas linguagens de programação aceitam também essas variáveis especiais do tipo *ponteiros*. Por exemplo, em C podemos criar `int *p=FF10AF`.
- Em geral, não precisamos saber qual é o endereço de memória de uma variável, mas é importante saber associar a um ponteiro. Por exemplo, a seguir é criada uma variável inteira `x=10` e em seguida um ponteiro `p` é associado a `x`, incluindo um prefixo `&`, para pegar o endereço de memória de `x` e associar a `p`, com `p = &x`. Para alterar o conteúdo de `p` (o mesmo de `x`), bastar incluir o prefixo `*`, ou seja, `*p = 15`. Ver também a figura a seguir para melhor visualizar operações com ponteiros.

```
int x = 10;
int *p;     // cria um ponteiro para um inteiro
p = &x;     // faz p apontar para o endereço de x
*p = 15;    // altera o conteúdo de p (e também de x)
printf("x=%d *p=%d", x, *p);
```

8.4 Alocação estática

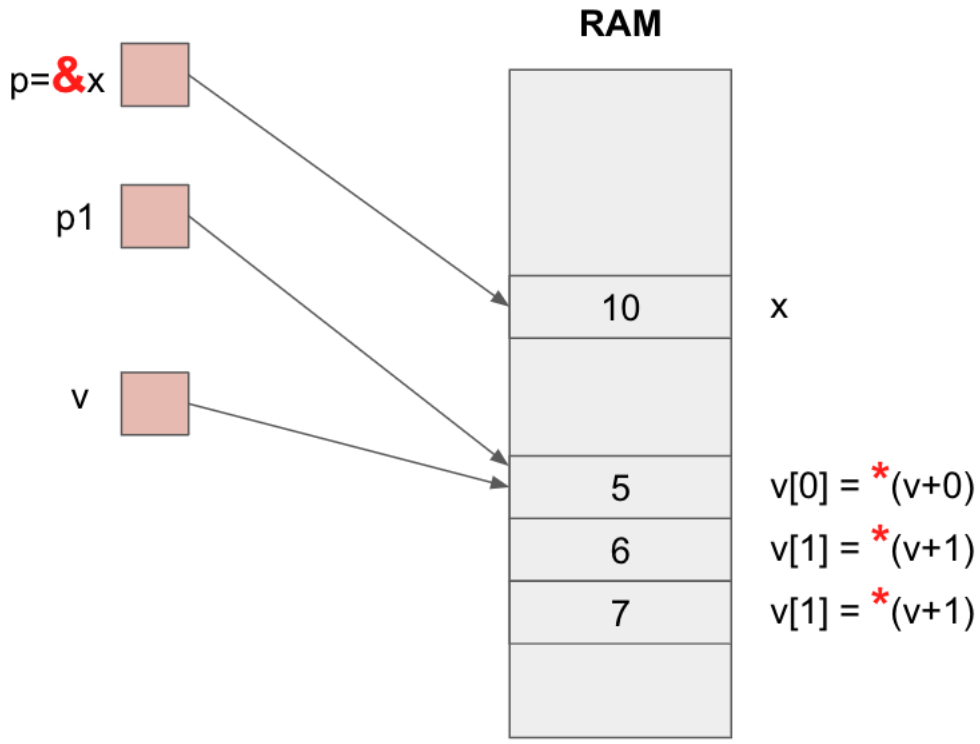
- A *alocação estática* ocorre em *tempo de compilação* a depender a linguagem de programação escolhida. Por exemplo, no exemplo a seguir, o vetor `v` é criado contendo 3 elementos inteiros. Não é possível incluir um quarto elemento nesse vetor quando executamos o progroma (ou seja, em *tempo de execução*).
- Ao criar vetores ou matrizes, estamos criando ponteiros para o primeiro elemento dessas estruturas. Por exemplo,

```
int v[3] = { 5,6,7 };
int* p1;
p1 = v; // observe que aqui não precisa usar &v
for (int i = 0; i < 3; i++)
  printf("%d %d %d\n", *(p1 + i), p1[i], v[i]); // *(p1+i) = p1[i] = v[i]
```

  - Assim, nesse exemplo é possível fazer operações de ponteiros (`p1 + i`) para acessar os endereços `p1, p1+1, p1+2`, $\$\cdots\$ Para a variável ponteiro do tipo `char`, `char *p` ocupa um byte. Para ints o 4 bytes e para doubles o 8 byte

- Para saber a quantidade de *bytes* de uma variável/tipo, usar a função `sizeof`.

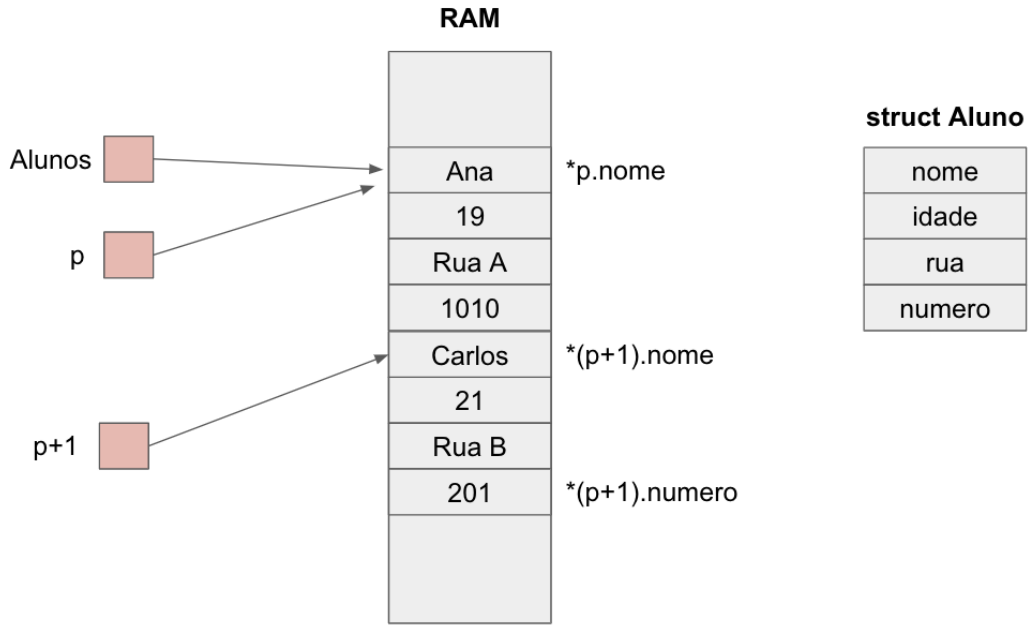
- Por exemplo, para uma variável do tipo `double`, basta fazer `sizeof (double)`, ou `sizeof(p1)`, onde `p1` foi definido no exemplo anterior.
- Também é possível calcular o tamanho de um vetor alocado estaticamente. Para o exemplo anterior, basta fazer: `sizeof(v) / sizeof(int)`.
- Esse cálculo é útil para saber o tamanho de um *array* porém tem algumas restriçoes, ver próxima seção.



- Também é possível fazer operações de ponteiros para *array* de `struct`. Ver um exemplo a seguir:

8.4.1 Exemplo 01 - Criar um registro de Aluno, com `scanf`

Exemplo para criar um *array* de `struct` **Aluno** contendo 4 atributos lidos do teclado com `scanf`. Associar um ponteiro para essa `struct`.



[ ]:
```
%%writefile cap8ex01.c
#include <stdio.h>
#include <string.h>

typedef struct {
  char nome[50];
  int idade;
  char rua[50];
  int numero;
} Aluno;

int main() {
  Aluno Alunos[2]; // instancia uma vetor do tipo Aluno
  Aluno *p; // cria um ponteiro para Aluno
  p = Alunos; // ATENÇÃO: associa p a array de alunos
  // ENTRADA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("Entre com os dados: nome, idade, rua, número:\n");
    fflush(stdin);
    fgets(p[i].nome, 50, stdin);
    scanf("%d", &p[i].idade);
    fflush(stdin);
    fgets(p[i].rua, 50, stdin);
    scanf("%d", &p[i].numero);
  }
  // SAÍDA DE DADOS
  for (int i = 0; i < 2; i++) {
    printf("nome: %s\nidade: %d\n", p[i].nome, p[i].idade);
```

```
    printf("rua: %s\nnúmero: %d\n", p[i].rua, p[i].numero);
  }
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap8ex01.c -o output
./output
```

8.5 Alocação dinâmica

- A alocação dinâmica ocorre quando não sabemos o tamanho de um *array* em tempo de compilação, sendo necessário alocar em tempo de execução.
- Até agora utilizamos alocação estática para definir o tamanho de um *array*. Por exemplo, para criar um *array* de alunos da UFABC, podemos criar um *array* "muito grande", por exemplo, contendo 100000 alunos.
- Nesse caso, temos dois problemas, se um dia tivermos mais que 100000 alunos, esse *array* não vai suportar.
- O outro problema é que em geral estamos desperdiçando memória RAM com as posição do *array* não utilizadas.
- Para resolver isso, utilizamos *alocação dinâmica*, inserindo ou retirando elementos do *array* conforme a demanda.
- Na linguagem C existem os seguintes comandos da biblioteca `stdlib.h`:
  - `malloc`
    - sintaxe para *array* de inteiros: `int * p = (int *) malloc( TAMANHO * sizeof(int) )`
    - comando utilizado para alocar um *array* de `TAMANHO` de inteiros.
    - O ponteiro `p` aponta para o primeiro elemento do *array*.
    - É possível definir um *array* para qualquer tipo de dado, não apenas para `int`, inclusive para `struct`.
    - Se `p = NULL` então não existe memória suficiente para a alocação.
  - `calloc`
    - sintaxe para *array* de inteiros: `int * p = (int *) calloc( TAMANHO, sizeof(int) )`
    - análogo ao `malloc`, porém:
      - o `calloc` possui dois argumentos;
      - inicializa todos os elementos com zeros;
      - logo, um pouco mais lento.
  - `realloc`
    - sintaxe para *array* de inteiros: `p = (int *) realloc( p, TAMANHO * sizeof(int) )`
    - realoca um *array*
  - `free`

* sempre que alocamos memória dinâmicamente devemos liberar a memória no final, com o camando `free(p)`.
* se isso não for feito, a memória será liberada somente quando o computador for desligado ou a aplicação encerrada (se o sistema operacional não tiver algum recurso de "coleta de lixo").

8.5.1 Exemplo de alocação estática

```
void main(){
    int TAMANHO = 0;
    scanf("%i", &TAMANHO);
    int v[TAMANHO]; // NÃO PODE ALTER O TAMANHO DE v APÓS CRIADO!!!
    // free(v); // LOGO, NÃO PODE USAR free!!!
}
```

Tamanho de um *array*

```
int main() {
  int v[5] = { 3,4,5,6,7 };

  printf("sizeof(v)=%ld\n", sizeof(v));
  printf("sizeof(int)=%ld\n", sizeof(int));
  printf("tamanho=%li\n", (int) sizeof(p) / sizeof(int));
  return 0;
}
```

Retorna tamanho CORRETO:

```
sizeof(v)=20
sizeof(int)=4
tamanho=5
```

Porém, aqui não retorna o esperado

```
void funcao(int* v) {
  printf("sizeof(v)=%ld\n", sizeof(v));
  printf("sizeof(int)=%ld\n", sizeof(int));
  printf("tamanho=%li\n", (int)sizeof(v) / sizeof(int));
}
int main() {
  int v[5] = { 3,4,5,6,7 };
  funcao(v);
  return 0;
}
```

Retorna tamanho INCORRETO:

```
sizeof(v)=8
sizeof(int)=4
tamanho=2
```

8.5.2 Exemplo de alocação dinâmica

```
void main(){
    int TAMANHO = 0;
    scanf("%i", &TAMANHO);
    int *v = (int *) malloc( TAMANHO * sizeof(int) );
    free(v); // LIBERAR A MEMÓRIA - S E M P R E !!!!
}
```

Não é possível retornar o tamanho de um *array* alocado dinamicamente

```
int main() {
  int* p = (int*)malloc(5 * sizeof(int));

  printf("sizeof(p)=%li\n", sizeof(p));
  printf("sizeof(int)=%li\n", sizeof(int));
  printf("tamanho=%li\n", (int) sizeof(p) / sizeof(int));
  return 0;
}
```

Retorna tamanho INCORRETO:

```
sizeof(p)=8
sizeof(int)=4
tamanho=2
```

8.6 Alocação dinâmica para *array* multidimensional

- É possível usar um *array* unidimensional para tratar matriz multidimensional. Por exemplo, para uma matriz de L linha e C colunas de inteiros, temos:

```
int * m = (int *) malloc( L*C*sizeof(int) );
for (int i=0; i < L; i++)
  for (int j=0; j < C; j++)
    scanf("%d", m + i*C +j); // m+i*C+j = &m[i*C+j]

// ou simplesmente
for (int i=0; i < L*C; i++)
    scanf("%d", m + i); // linha=i/C e coluna=i%C
```

- Para utilizarmos matrizes com alocação dinâmica é interessante trabalharmos com *ponteiro para ponteiro*:
  - primeiro alocamos memória para as linhas (L) da matriz:
    - `int **m = (int**) malloc( L * sizeof(int *) );`
  - em seguida alocamos memória para cada linha da matriz (colunas C):

```
for (int i=0; i < L; i++)
 m[i] = (int*) malloc( C * sizeof(int *) ); // m[i] = *(m+i)
```

  - para popular a matriz - opção 1:

```
for (int i=0; i < L; i++)
  for (int j=0; j < C; j++)
    scanf("%d", &m[i][j]); // &m[i][j] = m + i*C + j
```

  - para popular a matriz - opção 2:

```
for (int i=0; i < L*C; i++)
    scanf("%d", m+i); // m+i = &m[i/C][i%C]
```

  - não esquecer de LIBERAR A MEMÓRIA:
    * primeiro o conteúdo de cada linha:

```
for (int i=0; i < L; i++)
free(m[i]); // m[i] = *(m+i)
```

    * depois toda a matriz:

```
free(m);
```

- Quais as vantagens de usar matriz como **ponteiro de ponteiro**?

8.7 Exemplo 01 - Ler/Escrever matriz com métodos (cap.6 - Matriz)

- Analogamente ao que foi feito no capítulo sobre vetores, onde alocamos os vetores em tempo de execução através de métodos, é possível usar modularização para melhorar a organização, manutenção e reaproveitamento de código.
- Aqui é apresentado um método `leiaMatriz` e `escrevaMatriz` genéricos
- Para entrada de dados, ou seja, inserir valores nos elementos alocados na memória para uma matriz.
- Além de saída de dados, para escrever a matriz, linha por linha.

Pseudocódigo Exemplo 01: Considere um algoritmo para: * Ler um inteiro `L` (linhas) representando o número de alunos, * Ler um inteiro `C` representando o número de avaliações. * Considere a primeira coluna o `RA` do aluno, assim `C=C+1`. * Criar uma matriz `m` com dimensões `LxC`. * Ler todos os elementos da matriz. * Escrever todos os elementos da matriz, formando a saída, linha por linha, por exemplo, para uma matriz com 2 alunos e 3 avaliações, escreva:

```
LISTA DE ALUNOS vs Avaliações:
1234 4 3 9
3456 6 4 8
```

```
Função inteiro m[][] leiaMatriz(inteiro L, inteiro C):
    Instanciar e alocar uma matriz m de Reais com L x C
    Para cada i, de i=0; até i<L; passo i=i+1 faça
        Para cada j, de j=0; até j<C; passo j=j+1 faça
            m[i,j] = leia("Digite um número inteiro:");

Função escrevaMatriz(inteiro m[][], inteiro L, inteiro C):
    Instanciar e alocar uma matriz m de Reais com L x C
    Para cada i, de i=0; até i<L; passo i=i+1 faça
        Para cada j, de j=0; até j<C; passo j=j+1 faça
            escreva(" ", m[i,j]);
        escreva("\n"); // pula linha

// PROGRAMA PRINCIPAL
```

```
// ENTRADAS
inteiro L = leia("Digite o numero de alunos:")
inteiro C = leia("Digite o numero de avaliações:")
C = C + 1  // a primeira coluna é o RA
Instanciar uma matriz m com L linhas e C colunas

m = leiaMatriz(L,C)
// PROCESSAMENTO: ?

// SAÍDA
escrevaMatriz(m)
```

Casos para Teste Moodle+VPL Para o professor criar uma atividade VPL no Moodle para este Exemplo 01, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=2
3
1234
4
3
9
3456
6
4
8
output=
LISTA DE ALUNOS vs Avaliações:
1234 4 3 9
3456 6 4 8
```

[ ]:
```c
%%writefile cap8ex02.c
#include <stdio.h>
#include<malloc.h>

int ** leiaMatriz(int L, int C) {
  int **m = (int **)malloc(L*sizeof(int*));
  for (int i = 0; i < L; i++) {
    m[i] = (int *)malloc(C * sizeof(int)); // for each row allocate C␣
 ↪ints
    for (int j = 0; j < C; j++)
      scanf("%d", &m[i][j]);
  }
  return m;
}
void free_matrix(int **m, int L) {
    for (int i = 0; i < L; i++)
```

```
        free(m[i]);
    free(m);
 }
void escrevaMatriz(int **m, int L, int C) {
  for (int i = 0; i < L; i++) {
   for (int j = 0; j < C; j++)
      printf("%d\t", m[i][j]);
   printf("\n");
  }
}
int main(void) {
  // ENTRADA DE DADOS
  int L, C, **m;   // variaveis de referência m
  printf("Digite o número de alunos: ");
  scanf("%d", &L);

  printf("Digite o número de avaliações: ");
  scanf("%d", &C);

  C = C + 1; // a primeira coluna é o RA do aluno

  printf("Digite os elementos da matriz");
  m = leiaMatriz(L,C);

  // PROCESSAMENTO ?

  // SAÍDA DE DADOS
  printf("\nLISTA DE ALUNOS vs Avaliações:\n");
  printf("RA ");
  for (int i = 0; i < C-1; i++)
    printf("\t%d",(i+1));

  printf("\n");
  escrevaMatriz(m,L,C);
  free_matrix(m,L); // liberar memória alocado com malloc
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap8ex02.c -o output
./output
```

Acessando elemento com * Outras forma de acessar elementos em matriz.

```
for (int i = 0; i < L; i++)
 for (int j = 0; j < C; j++) {
    printf("%d\t", m[i][j]); // ou
    printf("%d\t", *(m + i*C + j));
```

```
   }
```

Ou utilizando apenas um laço para varrar uma matriz:

```
  for (int i = 0; i < L*C; i++) {
      printf("%d\t", m[i/C][i%C]); // ou
      printf("%d\t", *(m + i));
   }
```

Matriz Multimensional

```
for (int k = 0; k < D; k++) // profundidade
  for (int i = 0; i < L; i++) // linha
   for (int j = 0; j < C; j++) { // coluna
      printf("%d\t", m[k][i][j]); // ou
      printf("%d\t", *(m + k*L*C + i*C + j));
   }
```

Ou utilizando apenas um laço para varrar uma matriz:

```
  for (int i = 0; i < D*L*C; i++) {
      d = i/(L*C);
      printf("%d\t", m[d][(d-i)/C][(d-i)%C]); // ou
      printf("%d\t", *(m + i));
   }
```

8.8 Tipo Abstrato de Dado (TAD) Lista

Quando criamos uma estrutura de dados e um conjunto de métodos para manipular essa estrutura, podemos dizer que estamos criando um **Tipo Abstrato de Dados (TAD)** (ou também chamado **Tipo de Dado Abstrato**). Esse conceito é também utilizado em Programação Orientada a Objetos, encapsutando também os métodos na estrutura. A seguir apresentamos dois TADs: **TAD Lista Estática** e **TAD Lista Encadeada**.

8.8.1 TAD Lista Estática

Ao manipular listas estáticas, em algumas situações (por exemplo, não altera muito o tamanho máximo `MAX_LISTA`), é interessante se definir uma estrutura de dados como segue:

```
#define MAX_LISTA 6
typedef struct {
  int tamanho;
  int conteudos[MAX_LISTA];
} Lista0; // AQUI Lista estática é Lista0 e lista0
```

Além dessa estrutura, muitos métodos podem ser úteis para a sua manipulação. Por exemplo:

```
Lista0* lista0_cria(void);
void lista0_free(Lista0* lista);
int lista0_tamanho(Lista0* lista);
```

```
int lista0_cheia(Lista0* lista);
int lista0_insere(Lista0* lista, int conteudo); // insere no final
void lista0_imprime(Lista0* lista);
int lista0_insere_inicio(Lista0* lista, int conteudo);
int lista0_busca(Lista0* lista, int conteudo); // indice 1a ocorrencia
int lista0_remove(Lista0* lista); // remove no final
int lista0_remove_inicio(Lista0* lista);
int lista0_troca(Lista0* lista, int i, int j);
int lista0_remove_conteudo(Lista0* lista, int conteudo); // remove 1o conteudo
```

Esses métodos estão implementados e disponíveis no arquivo `myLista.c`, no GitHub, juntamente com o seu uso no programa `cap8.part1ex03.c`: https://github.com/fzampirolli/codigosPE/tree/master/cap8

Como esses métodos são aplicações dos conceitos já abordados anteriormente (capítulos 5-vetor, 6-matriz e 7-struct), vamos focar o restante deste capítulo em **Listas Encadeadas**.

8.8.2 TAD Lista Encadeada (ou dinâmica)

Uma vantagem de trabalhar com **Lista Encadada**, em comparação com as listas estáticas, é que os elementos da lista são inseridos/removidos em tempo de execução.

Com isso, a princípio não precisamos definir um tamanho da lista (número de elementos), muito menos o tamanho máximo, necessários em listas estáticas.

Cada elemento de uma lista pode ser definido contendo um conteúdo (por exemplo, do tipo inteiro) e um ponteiro para o próximo elemento da lista, como segue:

```
typedef struct TCelula Celula;
struct TCelula {
  int conteudo;
  Celula* prox;
};
```

Uma célula `c` pode ser declarada e manipulada assim:

```
Celula c;
c.conteudo = 10;
c.prox = NULL;
```

Um ponteiro `p` para uma célula pode ser definido e manipulado assim:

```
Celula *p  = (Celula*)malloc(sizeof(Celula));
*p->conteudo = 10;
*p->pro = NULL;
```

Uma lista encadeada é um conjunto de células interlicadas e pode ser acessada pelo endereço da primeira célula da lista.

Existem várias formas de criar uma lista e de implementar métodos para manipular listas encadeadas.

Por exemplo, utilizando ponteiro para ponteiro, lista com "cabeça" e simplemente, criando um ponteiro para o início da lista, ver figura a seguir.

Ponteiro para ponteiro

```
typedef Celula *Lista; // lista é ponteiro para ponteiro
Lista **lista = (Lista *) malloc(sizeof(Lista));
*lista = NULL; // lista vazia
```

Incluir um primeiro elemento da lista:

```
Celula* no = (Celula*)malloc(sizeof(Celula));
no->conteudo = conteudo;
no->prox = *lista;
*lista = no;
```

Cabeça

```
Celula* cabeca = (Celula*)malloc(sizeof(Celula));
cabeca->prox = NULL; // lista vazia
```

Incluir um primeiro elemento da lista:

```
Celula* no = (Celula*)malloc(sizeof(Celula));
no->conteudo = conteudo;
no->prox = NULL;
cabeca->prox = no;
```

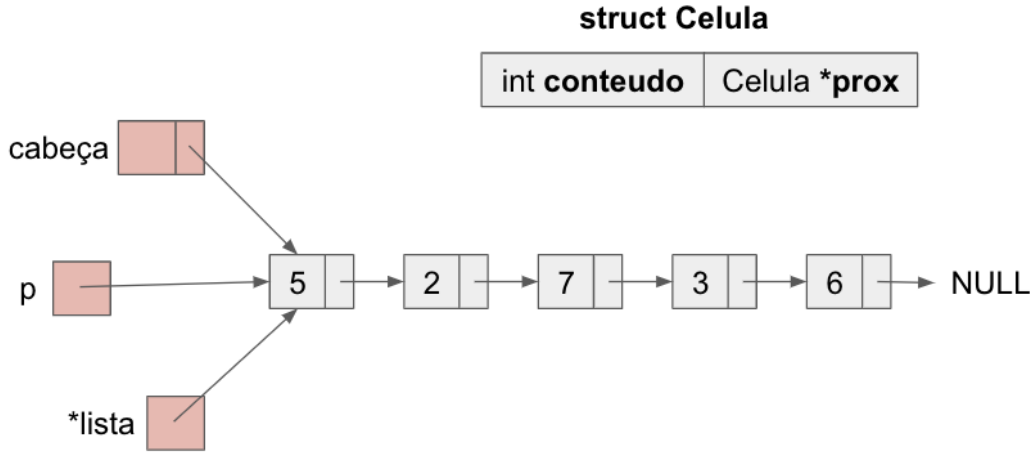
Ponteiro

```
Celula* p = NULL ; // lista vazia
```

Incluir um primeiro elemento da lista:

```
Celula* no = (Celula*)malloc(sizeof(Celula));
no->conteudo = conteudo;
no->prox = NULL;
p = no;
```

Para simplificar alguns métodos, vamos apresentar a seguir algumas implementações para manipular lista com cabeça (o primeiro elemento, a cabeça, tem o campo `prox` apontando para o primeiro elemento da lista):



8.8.3 Exemplo 03 - Criar e manipular lista encadeada

[ ]:
```
%%writefile myLista.h
#include <stdio.h>
#include <stdlib.h>

typedef struct TCelula Celula;
struct TCelula {
  int conteudo;
  Celula* prox;
};
Celula* lista_cria(void);
void lista_insere(Celula* p, int conteudo);
void lista_imprime(Celula* cabeca);
void lista_free(Celula* cabeca);
void lista_remove(Celula* p);
void lista_busca_remove(Celula* cabeca, int conteudo);
void lista_busca_insere(Celula* cabeca, int busca, int novo);
```

[ ]:
```
%%writefile myLista.c
#include "myLista.h"
Celula* lista_cria(void) {
  Celula* cabeca = (Celula*)malloc(sizeof(Celula));
  if (cabeca == NULL) {
    printf("ERRO: sem memoria\n");
    exit(1);
  }
  cabeca->prox = NULL;
  return cabeca;
}
void lista_insere(Celula* p, int conteudo) {
  // insere numa posicao p qualquer da lista
  if (p == NULL) exit(1);
  Celula* novo = (Celula*)malloc(sizeof(Celula));
  novo->conteudo = conteudo;
  novo->prox = p->prox;
  p->prox = novo;
}
void lista_free(Celula* cabeca) {
  if (cabeca == NULL) exit(1);
  Celula* aux = cabeca->prox;
  while (aux != NULL) {
    Celula* no = aux;
    aux = aux->prox;
    free(no);
```

```
  }
  free(cabeca);
}
void lista_imprime(Celula* cabeca) {
  Celula* p;
  for (p = cabeca->prox; p != NULL; p = p->prox)
    printf("%d ", p->conteudo);
  printf("\n");
}
void lista_remove(Celula* p) { // remove p->prox
  if (p == NULL) exit(1);
  Celula* no;
  no = p->prox;
  p->prox = no->prox;
  free(no);
}
void lista_busca_remove(Celula* cabeca, int conteudo) {
  if (cabeca == NULL) exit(1);
  Celula* no, * antes;
  antes = cabeca;
  no = antes->prox;
  while (no != NULL && no->conteudo != conteudo) {
    antes = no;
    no = no->prox;
  }
  lista_remove(antes);
}
void lista_busca_insere(Celula* cabeca, int busca, int novo) {
  // insere antes da primeira ocorrencia de busca ou no final
  if (cabeca == NULL) exit(1);
  Celula* no, * antes;
  antes = cabeca;
  no = antes->prox;
  while (no != NULL && no->conteudo != busca) {
    antes = no;
    no = no->prox;
  }
  lista_insere(antes, novo);
}
```

[ ]:
```
%%writefile cap8ex03.c
#include "myLista.h"
int main() {
  Celula* cabeca = lista_cria();
  for (int i = 0; i < 6; i++)
    lista_insere(cabeca, i + 1);
  lista_imprime(cabeca);
```

```
  lista_remove(cabeca);
  lista_imprime(cabeca);
  lista_busca_remove(cabeca, 3);
  lista_imprime(cabeca);
  lista_busca_insere(cabeca, 2, 8);
  lista_imprime(cabeca);
  lista_free(cabeca);
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 myLista.c cap8ex03.c -o output
./output
```

DESAFIO: Desenhar o passo-a-passo de cada método no programa anterior, como feito da figura acima.

8.9 Tipo Abstrato de Dado (TAD) Fila

[Ref. Notas do prof. Paulo Feofiloff]

Semelhante ao TAD Lista, é possível criar **TAD Fila**, tendo que gerenciar o primeiro (P) e o último (U) elementos dessa estrutura. Ou seja, filas são casos particulares de listas. No caso de **Fila Estática**, também é necessário gerenciar o tamanho (N) da fila.

Observe então que TAD Fila não permite inserir/remover elementos do meio da estrutura, como ocorrem em TAD Lista.

Semelhante a uma fila (honesta ou sem prioridades) de caixa de banco/supermercado, o primeiro a entrar deverá ser também o primeiro a sair (*FIFO = First-In-First-Out*).

Para facilitar, é possível criar variáveis globais para gerencia esses elementos (P, U e N), ou também, criar uma outra estrutura de dados contendo essas informações, a gosto do desenvolvedor.

8.9.1 TAD Fila Estática

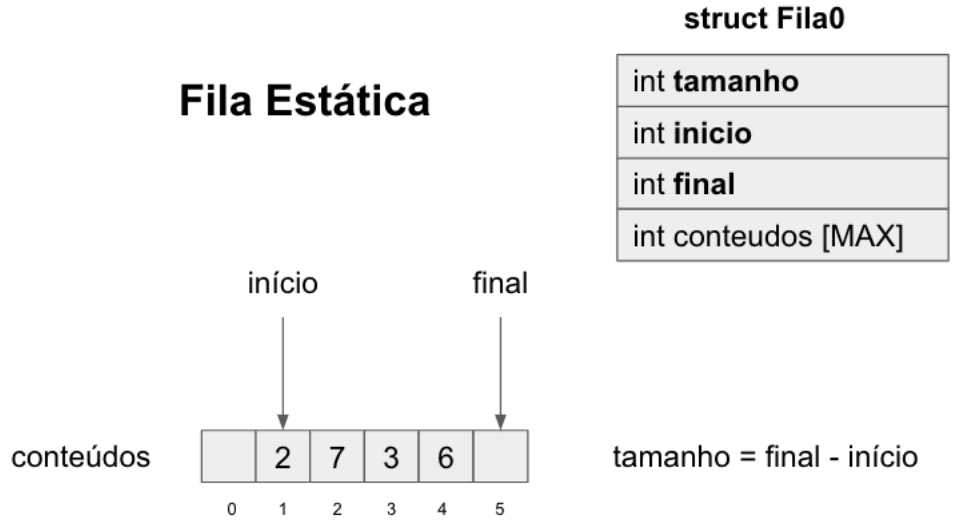
Ao manipular dados, em algumas situações (por exemplo, quando não altera muito o tamanho máximo `MAX`), é interessante se definir uma estrutura de dados estática, como segue:

```
#define MAX 6
typedef struct {
  int tamanho;
  int inicio;
  int final;
  int conteudos[MAX];
} Fila0; // AQUI Fila Estática é Fila0 e fila0
```

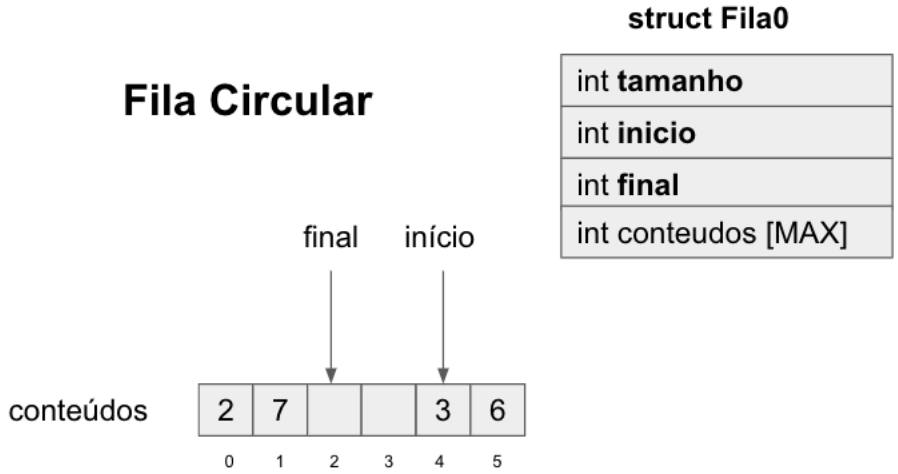
Além dessa estrutura, muitos métodos podem ser úteis para a sua manipulação. Por exemplo:

```
Fila0* fila0_cria(void);
void fila0_free(Fila0* fila);
int fila0_tamanho(Fila0* fila);
int fila0_cheia(Fila0* fila);
void fila0_imprime(Fila0* fila);
int fila0_insere(Fila0* fila, int conteudo); // insere no final
int fila0_remove(Fila0* fila); // remove no inicio
```

Esses métodos estão implementados e disponíveis no arquivo `myFila.c`, no GitHub, juntamente com o seu uso no programa `cap8.part1ex06.c`: https://github.com/fzampirolli/codigosPE/tree/master/cap8



Observe que o `tamanho` da fila é facilmente calculado, não sendo essencial para a `struct`. Observe também que na Fila Estática, onde só é possível inserir no final (e remover no início), fica rapidamente cheia (vazia), mas com várias posições do vetor sem conter elementos da fila. Uma solução é manipular uma fila circular, como ilustrado na figura a seguir. Outra solução é ir dobrando o tamanho da fila com o comando `realloc` para `2*MAX`, `4*MAX`, $\cdots$.



8.9.2 TAD Fila Encadeada

Uma vantagem de trabalhar com **Fila Encadada**, em comparação com as filas estáticas, é que os elementos da fila são inseridos/removidos em tempo de execução.

Com isso, a princípio não precisamos definir um tamanho da fila (número de elementos), muito menos o tamanho máximo, necessários em filas estáticas.

Cada elemento de uma fila pode ser definido contendo um conteúdo (por exemplo, do tipo inteiro) e um ponteiro para o próximo elemento da fila, idêntico ao utilizado em listas, como segue:

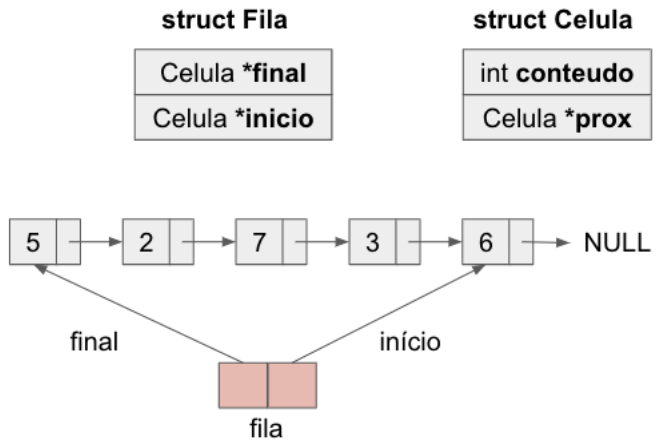
```
typedef struct TCelula Celula;
struct TCelula {
  int conteudo;
  Celula* prox;
};
```

Um ponteiro `p` para uma célula pode ser definido e manipulado assim:

```
Celula *p  = (Celula*)malloc(sizeof(Celula));
*p->conteudo = 10;
*p->pro = NULL;
```

Uma fila encadeada é um conjunto de células interlicadas e pode ser acessada pelo endereço da primeira célula da fila `final`. Lembrando que se insere no final e se remone no início da fila. Assim, é necessário um outro endereço para o `inicio` da fila. Podemos encapsular esses ponteiros em uma nova `scruct`, ver figura a seguir.

```
typedef struct TFila Fila;
struct TFila {
  Celula* inicio;
  Celula* final;
};
```



8.9.3 Exemplo 04 - Criar e manipular fila encadeada

[1]:
```
%%writefile myFila.h
#include <stdio.h>
#include <stdlib.h>

#define MAX 6
typedef struct TCelula Celula;
struct TCelula {
  int conteudo;
  Celula* prox;
};
typedef struct TFila Fila;
struct TFila {
  Celula* inicio;
  Celula* final;
};
//////////////////// Versão ***SEM*** PONTEIRO DE PONTEIRO
Fila* fila_cria(void);
void fila_free(Fila* fila);
void fila_imprime(Fila* fila);
void fila_insere(Fila* fila, int conteudo);
void fila_remove(Fila* fila);
```

[2]:
```
%%writefile myFila.c
#include "myFila.h"
//////////////////// Versão 2 - ***SEM*** PONTEIRO DE PONTEIRO
Fila* fila_cria(void) {
  Fila* f = (Fila*)malloc(sizeof(Fila));
  f->inicio = f->final = NULL;
  return f;
}
void fila_insere(Fila* fila, int conteudo) {
  Celula* novo = (Celula*)malloc(sizeof(Celula));
  novo->conteudo = conteudo;
  novo->prox = fila->final;
  if (fila->inicio == NULL)
    fila->inicio = novo;
  fila->final = novo;
}
void fila_imprime(Fila* fila) {
  for (Celula* p = fila->final; p != NULL; p = p->prox)
    printf("%d ", p->conteudo);
  printf("\n");
}
void fila_free(Fila* fila) {
  if (fila == NULL) exit(1);
  Celula* aux = fila->final;
```

```
  while (aux != fila->inicio) {
    Celula* no = aux;
    aux = aux->prox;
    free(no);
  }
  free(fila);
}
void fila_remove(Fila* fila) { // remove p->prox
  Celula* ant = fila->final;
  while (ant->prox != fila->inicio)
    ant = ant->prox;
  ant->prox = NULL;
  free(fila->inicio);
  fila->inicio = ant;
}
```

[3]:
```
%%writefile cap8ex04.c
#include "myFila.h"
int main() {
  Fila* fila = fila_cria();

  for (int i = 0; i < MAX; i++)
    fila_insere(fila, i + 1);
  fila_imprime(fila);
  fila_remove(fila);
  fila_imprime(fila);
  fila_remove(fila);
  fila_imprime(fila);
  fila_free(fila);
  return 0;
}
```

[5]:
```
%%shell
gcc -Wall -std=c99 myFila.c cap8ex04.c -o output
./output
```

DESAFIO: Desenhar o passo-a-passo de cada método no programa anterior, como feito da figura acima.

8.10 Tipo Abstrato de Dado (TAD) Pilha

[Ref. Notas do prof. Paulo Feofiloff]

Semelhante ao TAD Lista, é possível criar **TAD Pilha**, tendo que gerenciar somente o último elemento dessa estrutura (`T=Topo` da pilha), inserindo ou removendo. Ou seja, pilhas são casos particulares de listas. No caso de **Pilha Estática**, também é necessário gerenciar o tamanho (N) da pilha.

Observe então que TAD Pilha não permite inserir/remover elementos do meio da estru-

tura, como ocorrem em TAD Lista.

Semelhante a uma pilha de pratos, o primeiro a entrar deverá ser o último a sair (*LIFO = Last-In-First-Out*).

Para facilitar, é possível criar variáveis globais para gerencia esses elementos (T e N), ou também, criar uma outra estrutura de dados contendo essas informações, a gosto do desenvolvedor.

8.10.1 TAD Pilha Estática

Ao manipular dados, em algumas situações (por exemplo, quando não altera muito o tamanho máximo `MAX`), é interessante se definir uma estrutura de dados estática, como segue:

```
#define MAX 6
typedef struct {
  int tamanho; // topo da pilha
  int conteudos[MAX];
} Pilha0; // AQUI pilha Estática é Pilha0 e pilha0
```

Além dessa estrutura, muitos métodos podem ser úteis para a sua manipulação. Por exemplo:

```
Pilha0* pilha0_cria(void);
void pilha0_free(Pilha0* pilha);
int pilha0_tamanho(Pilha0* pilha);
int pilha0_cheia(Pilha0* pilha);
void pilha0_imprime(Pilha0* pilha);
int pilha0_insere(Pilha0* pilha, int conteudo); // insere no topo
int pilha0_remove(Pilha0* pilha); // remove no topo
```

Esses métodos estão implementados e disponíveis no arquivo `myPilha.c`, no GitHub, juntamente com o seu uso no programa `cap8.part1ex06.c`: https://github.com/fzampirolli/codigosPE/tree/master/cap8

Observe que na Pilha Estática, onde só é possível inserir e remover no final, fica rapidamente cheia, mas com várias posições do vetor sem conter elementos da pilha. Uma solução é ir dobrando o tamanho da pilha com o comando `realloc` para `2*MAX`, `4*MAX`, $\cdots$.

8.10.2 TAD Pilha Encadeada

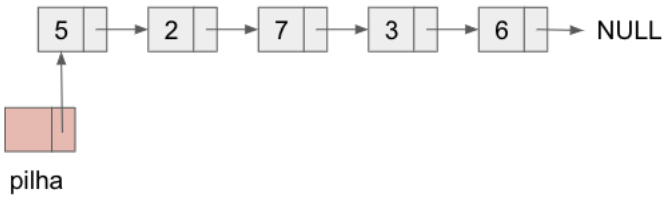
Uma vantagem de trabalhar com **Pilha Encadada**, em comparação com as pilhas estáticas, é que os elementos da pilha são inseridos/removidos em tempo de execução.

Com isso, a princípio não precisamos definir um tamanho da pilha (número de elementos), muito menos o tamanho máximo, necessários em pilhas estáticas.

Cada elemento de uma pilha pode ser definido contendo um conteúdo (por exemplo, do tipo inteiro) e um ponteiro para o próximo elemento da pilha, idêntico ao utilizado em listas, como segue:

```
typedef struct TCelula Celula;
struct TCelula {
  int conteudo;
  Celula* prox;
};
```

Uma pilha encadeada é um conjunto de células interlicadas e pode ser acessada pelo endereço da primeira célula da `pilha` (aqui será utilizada uma célula "cabeça" apontando para to topo da pilha). Lembrando que se insere no final e se remove também no final da pilha.



8.10.3 Exemplo 04 - Criar e manipular pilha encadeada

[1]:
```
%%writefile myPilha.h
#include <stdio.h>
#include <stdlib.h>

#define MAX 6
typedef struct TCelula Celula;
struct TCelula {
  int conteudo;
  Celula* prox;
};
Celula* Pilha;

//////////////////// Versão ***SEM*** PONTEIRO DE PONTEIRO
Celula* pilha_cria(void);
void pilha_free(Celula* pilha);
void pilha_imprime(Celula* pilha);
void pilha_insere(Celula* pilha, int conteudo);
void pilha_remove(Celula* pilha);
```

[2]:
```
%%writefile myPilha.c
#include "myPilha.h"
//////////////////// Versão 2 - ***SEM*** PONTEIRO DE PONTEIRO
// ref. https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html
Celula* pilha_cria(void) {
```

```
  Celula* cabeca = (Celula*)malloc(sizeof(Celula));
  if (cabeca == NULL) {
    printf("ERRO: sem memoria\n");
    exit(1);
  }
  cabeca->prox = NULL;
  return cabeca;
}
void pilha_insere(Celula* pilha, int conteudo) {
  if (pilha == NULL) exit(1);
  Celula* novo = (Celula*)malloc(sizeof(Celula));
  novo->conteudo = conteudo;
  novo->prox = pilha->prox;
  pilha->prox = novo;
}
void pilha_imprime(Celula* pilha) {
  for (Celula* p = pilha->prox; p != NULL; p = p->prox)
    printf("%d ", p->conteudo);
  printf("\n");
}
void pilha_free(Celula* pilha) {
  if (pilha == NULL) exit(1);
  Celula* aux = pilha->prox;
  while (aux != NULL) {
    Celula* no = aux;
    aux = aux->prox;
    free(no);
  }
  free(pilha);
}
void pilha_remove(Celula* pilha) { // remove p->prox
  if (pilha == NULL) exit(1);
  Celula* no;
  no = pilha->prox;
  pilha->prox = no->prox;
  free(no);
}
```

[3]:
```
%%writefile cap8ex05.c
#include "myPilha.h"
int main() {
  Celula* pilha = pilha_cria();

  for (int i = 0; i < MAX; i++)
    pilha_insere(pilha, i + 1);
  pilha_imprime(pilha);
  pilha_remove(pilha);
```

```
  pilha_imprime(pilha);
  pilha_remove(pilha);
  pilha_imprime(pilha);
  pilha_free(pilha);
  return 0;
}
```

[5]:
```
%%shell
gcc -Wall -std=c99 myPilha.c cap8ex05.c -o output
./output
```

DESAFIO: Desenhar o passo-a-passo de cada método no programa anterior, como feito da figura acima.

8.11 Exercícios

Ver notebook Colab nos arquivos `cap8.partX.lab.*.ipynb` (`X` ∈ `[2,3,4,5]` e `*` é a extensão da linguagem), utilizando alguma linguagem de programação de sua preferência, onganizadas em subpastas contidas em `"gen"`, na pasta do Google Drive colabs.

8.12 Revisão deste capítulo

- Introdução
- Alocação estática
- Alocação dinâmica
- Alocação dinâmica para *array* multidimensional
- Tipo Abstrato de Dados (TAD) Lista
- Tipo Abstrato de Dados (TAD) Fila
- Tipo Abstrato de Dados (TAD) Pilha
- Exercícios
- Revisão deste capítulo de Vetores