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

17 de dezembro 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,

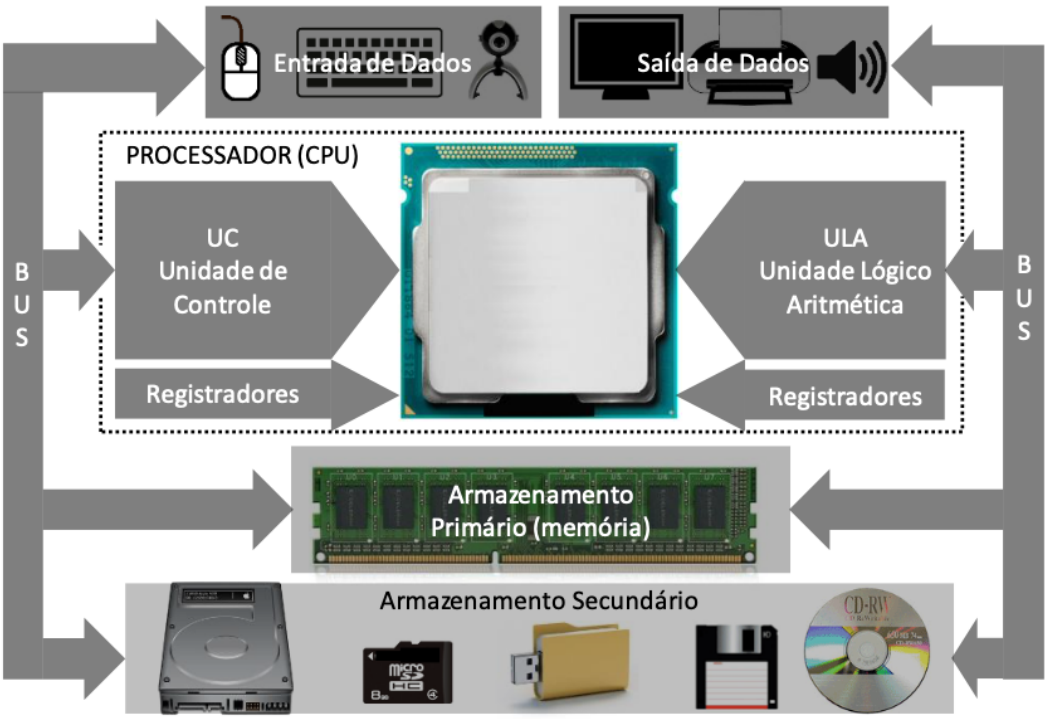
https://replit.com/, https://www.jdoodle.com
- É possível também rodar códigos no seu celular, utilizando navegadores acessando esses links.
- Esse arquivo com extensão `ipynb` pode ser editado também no Jupyter Notebook - https://jupyter.org.

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:

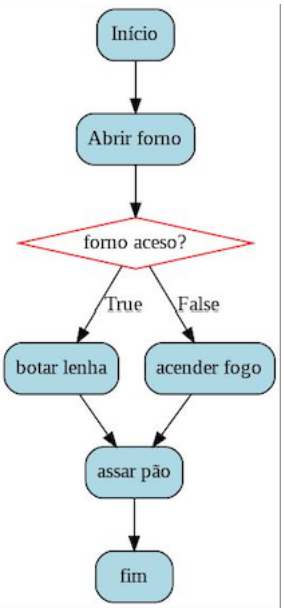
```
leia(a,b,c)
calcular delta = b**2-4*a*c
se delta < 0:
   escreva(x não possui raízes reais)
   fim do programa
se delta = 0:
   escreva(x possui duas raízes reais idênticas)
se delta > 0:
   escreva(x possui duas raízes reais e distintas)
calcule x=a*x*x+b*x+c
escreva(x)
```

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.

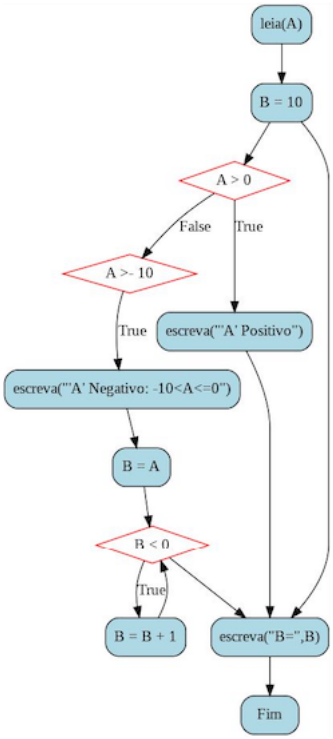
[ ]:
```
# instalando algumas bibliotecas no servidor do Colab (Linux)
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
# 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.

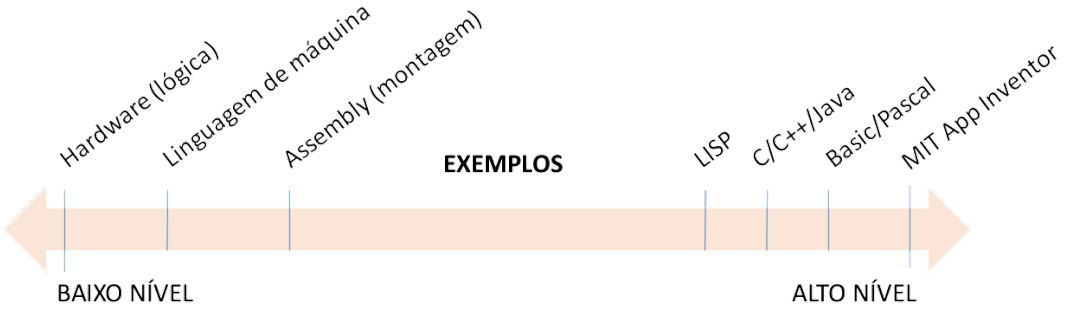
1.5 Conceitos de Linguagens de Programação

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