

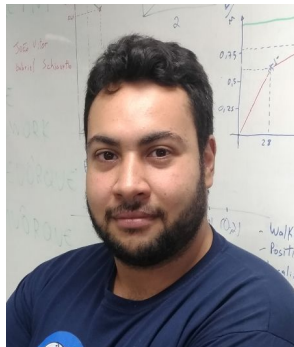
# CCP130

# Desenvolvimento de Algoritmos

Prof. Danilo H. Perico



# Teoria e Laboratório



**Prof. Isaac Jesus da  
Silva**

isaacjesus@fei.edu.br

- Técnico em Mecatrônica (SENAI Armando Arruda Pereira)
- Engenheiro Eletricista com Ênfase em Eletrônica (FEI)
- Mestre em Inteligência Artificial Aplicada à Automação (FEI)
- Doutor em Inteligência Artificial Aplicada à Automação (FEI)
- Coordenador da Categoria de Corrida de Robôs Humanoides na RoboCup Brasil

# Teoria e Laboratório



Prof. André Luiz Perin

andreperin@fei.edu.br

- Técnico em Mecânica (Etec Lauro Gomes)
- Engenheiro da Computação (USJT)
- Especialista em Engenharia de Software (USJT)
- Mestre em Engenharia Elétrica - Microeletrônica (FEI)
- Doutor em Engenharia Elétrica - Microeletrônica (FEI)
- Estuda dispositivos CMOS submetidos à radiação e campo magnético

# Teoria e Laboratório



**Prof. Danilo Hernani  
Perico**

dperico@fei.edu.br

- Técnico em Informática (Etec Lauro Gomes)
- Engenheiro Eletricista com Ênfase em Eletrônica (FEI)
- Mestre em Inteligência Artificial Aplicada à Automação (FEI)
- Doutor em Inteligência Artificial Aplicada à Automação (FEI)
- Secretário do IEEE Robotics & Automation Systems (IEEE RAS) - Seção Sul Brasil
- Coordenador da Categoria de Futebol de Robôs Humanoides na RoboCup Brasil

# Teoria e Laboratório



**Prof. Daniel Rodrigues  
da Silva**

drsilva@fei.edu.br

- Bacharel em Matemática Aplicada (USP)
- Doutor em Ciências - Matemática Aplicada (USP)
- Pós-Doc em Matemática (USP)
- Experiência na área de Matemática Aplicada com Equações Diferenciais, Simulação Numérica e Inferência Estatística



# Desenvolvimento de Algoritmos

Objetivos da disciplina:

- Desenvolver programas aplicando os conceitos estudados de **modularização** e **estruturas básicas de dados** ( vetores, matrizes e registros) e suas aplicações.
- Discutir e analisar soluções correspondentes a **problemas computacionais** envolvendo os conceitos estudados e propor soluções adequadas para eles
- **Implementar os programas** de computador correspondentes às soluções desses problemas.



# Desenvolvimento de Algoritmos

Finalidade da disciplina:

- Aprender conceitos avançados da **linguagem C**
- **Motivar** o aluno a **pensar** em soluções algorítmicas de problemas e implementá-las

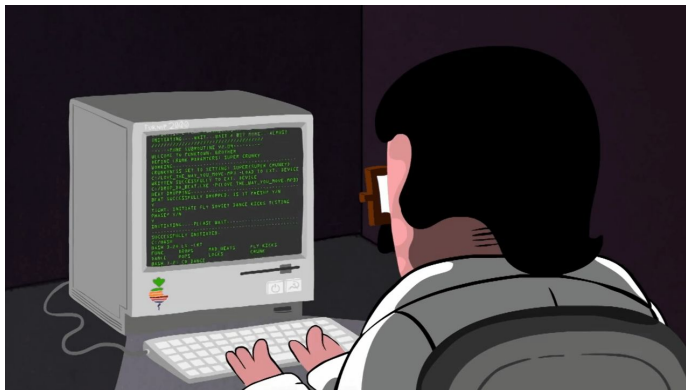


# Desenvolvimento de Algoritmos

- Existem 3 formas para aprender a programar



# Desenvolvimento de Algoritmos



1<sup>a</sup> - Programando!

# Desenvolvimento de Algoritmos



## 2ª - Programando!

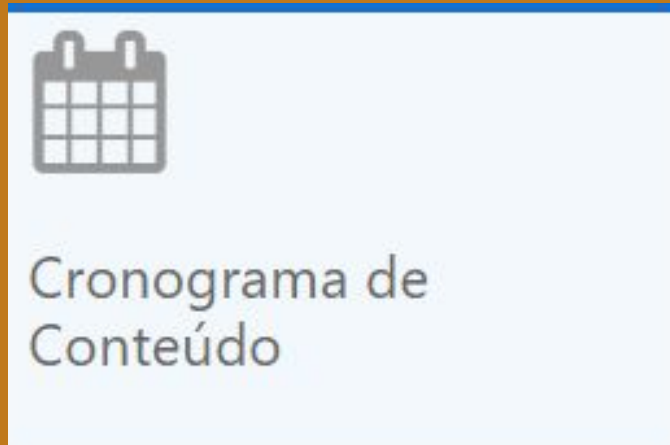
# Desenvolvimento de Algoritmos



## 3ª - Programando!

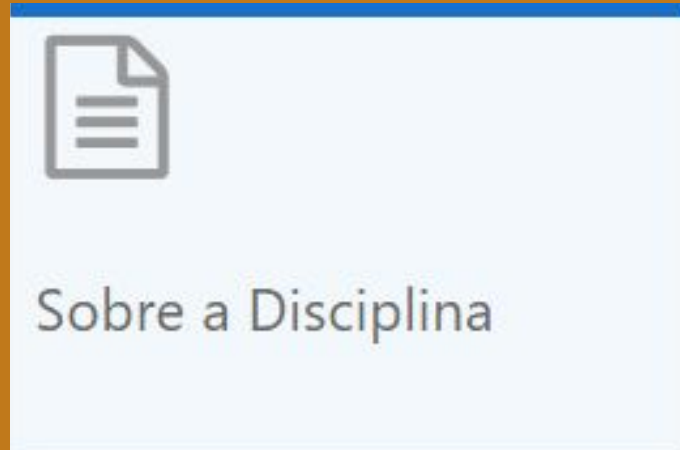
# Conteúdo Programático

- No Moodle:
  - **Cronograma de Conteúdo**

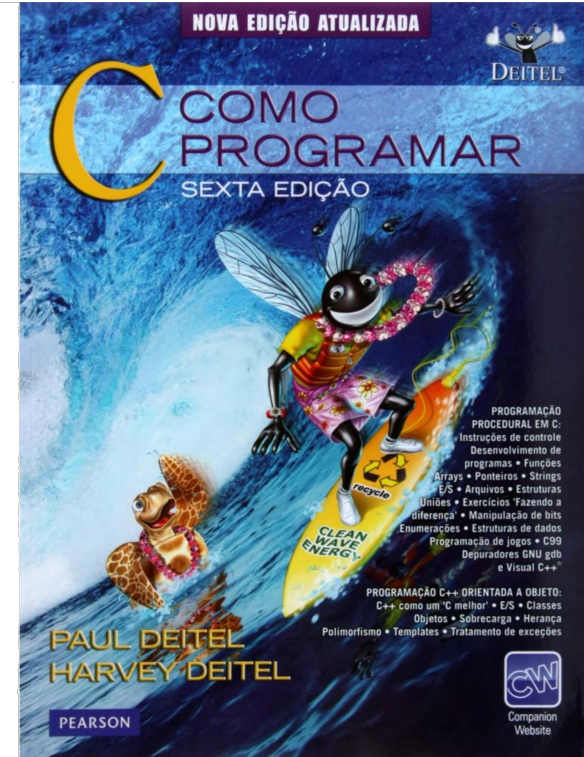
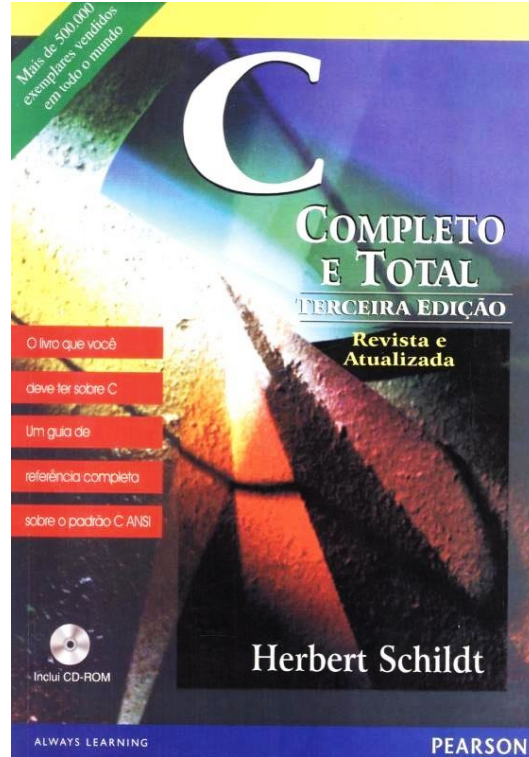
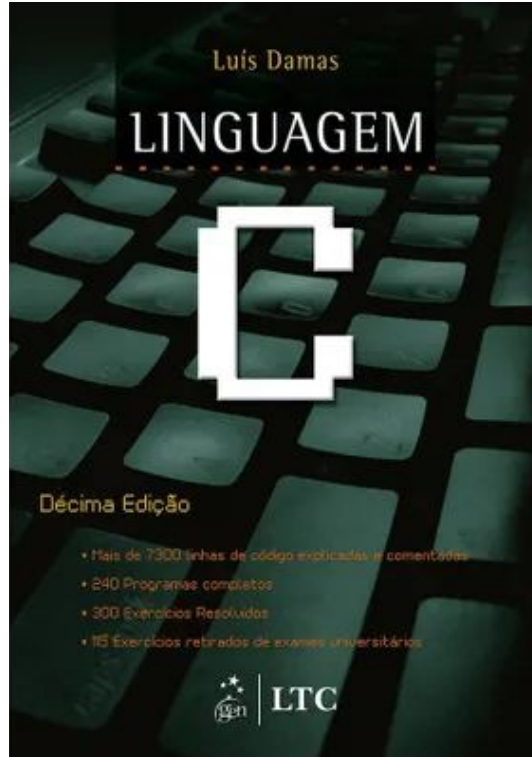


# Critério de Avaliação

- No Moodle:
  - **Sobre a Disciplina**



# Bibliografia Básica



# Introdução: Linguagem C



O que é a linguagem C?



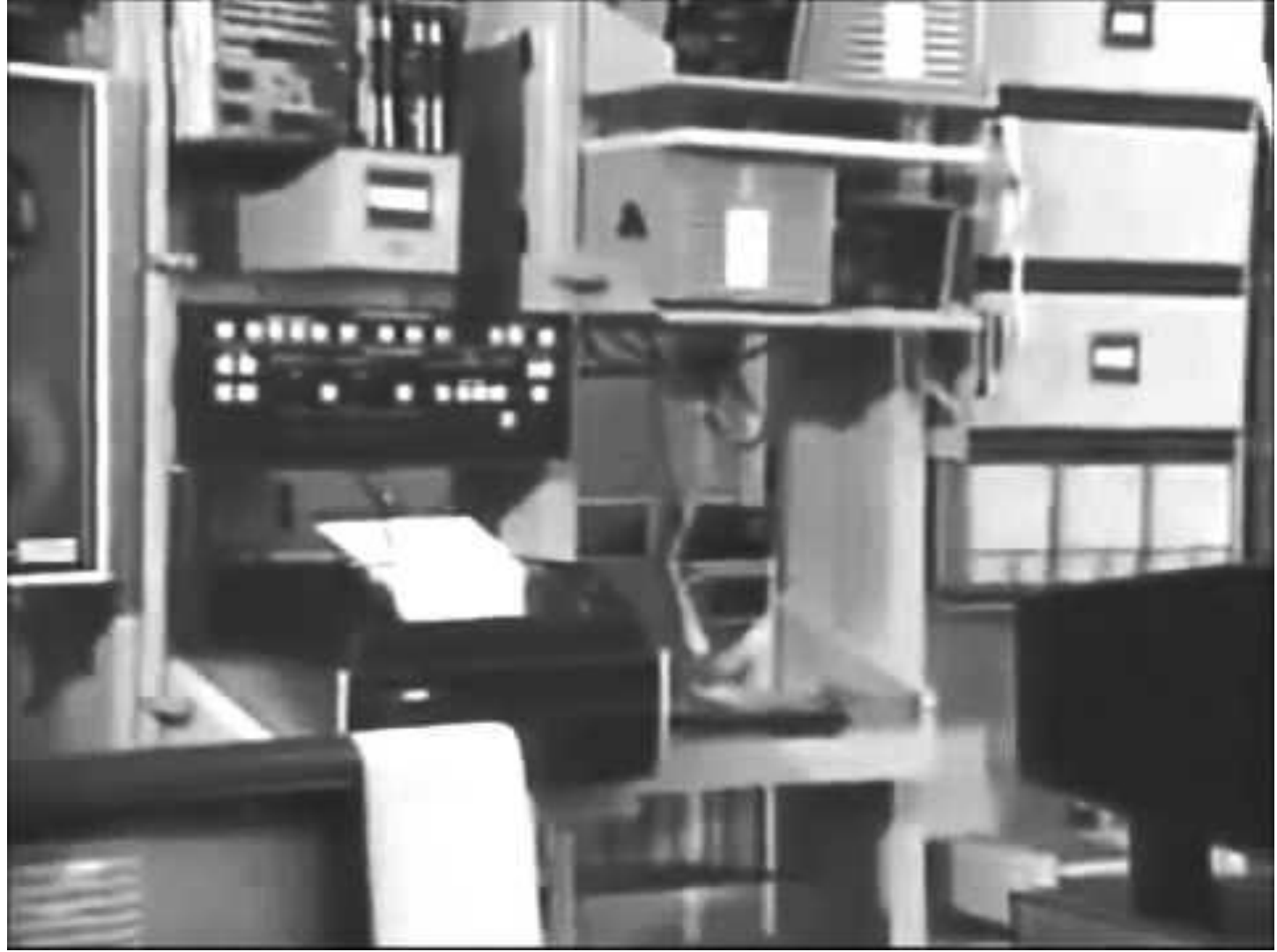




## O que é a linguagem C?

- A linguagem **C** foi criada em **1972** por **Dennis Ritchie** no **Bell Laboratories**.
- **C** foi criada para escrita de um Sistema Operacional - o **Unix**.
- **C** é uma linguagem de **programação estruturada** e **compilada**
- **C** é derivada de duas outras linguagens: **Algol 68** e **BCPL**







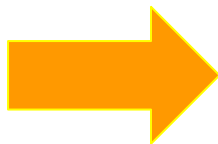
## Compilação

- Processo de tradução de um programa em uma linguagem de **programação textual** (como o C), que pode ser compreendida por um ser humano, para uma **linguagem de máquina**, que é específica para um processador e sistema operacional.
- Linguagens compiladas executam o código diretamente no sistema operacional ou processador, portanto são **mais rápidas**

# Compilação

## Código em C

```
1  /*****
2  * Programa de teste
3  * AUTOR: Danilo Perico
4  * DATA: 05/02/2020
5  *****/
6
7  #include <stdio.h>
8
9  int main(void) {
10     printf("Hello World\n");
11     return 0;
12 }
```

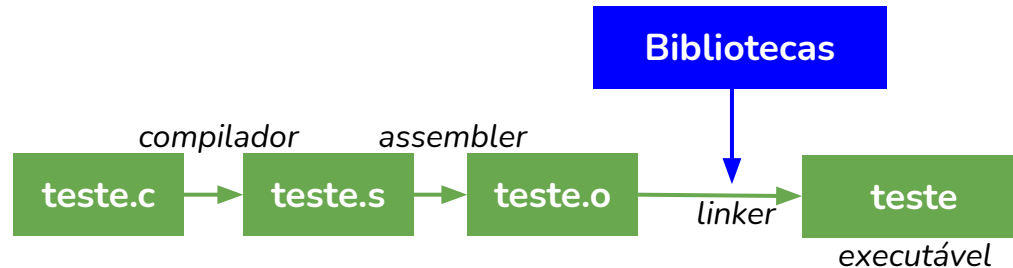


## Código de máquina

```
1  .file "teste.c"
2  .text
3  .section .rodata
4  .LC0:
5  .string "Hello World\007"
6  .text
7  .globl main
8  .type main, @function
9  main:
10 .LFB0:
11 .cfi_startproc
12 endbr64
13 pushq %rbp
14 .cfi_def_cfa_offset 16
15 .cfi_offset 6, -16
16 movq %rsp, %rbp
17 .cfi_def_cfa_register 6
18 leaq .LC0(%rip), %rdi
19 movl $0, %eax
20 call printf@PLT
21 movl $0, %eax
22 popq %rbp
23 .cfi_def_cfa 7, 8
24 ret
25 .cfi_endproc
26 .LFE0:
27 .size main, .-main
28 .ident "GCC: (Ubuntu 9.2.1-9ubuntu2) 9.2.1 20191008"
29 .section .note.GNU-stack,"",@progbits
30 .section .note.gnu.property,"a"
31 .align 8
32 .long 1f - 0f
33 .long 4f - 1f
34 .long 5
35 0:
36 .string "GNU"
37 1:
38 .align 8
39 .long 0xc0000002
```

# Compilação

- Compilação é realizada em três fases:
  - **Pré-processamento:** substitui/inclui partes do código
  - **Compilação:** produz a linguagem de máquina
  - **Ligação:** as bibliotecas são incluídas no executável





## Por que usar a linguagem C?

- C é uma linguagem potente e extremamente flexível
- É rápida em tempo de execução
- É popular
- É uma linguagem estável
- É muito utilizada

# Por que usar a linguagem C?



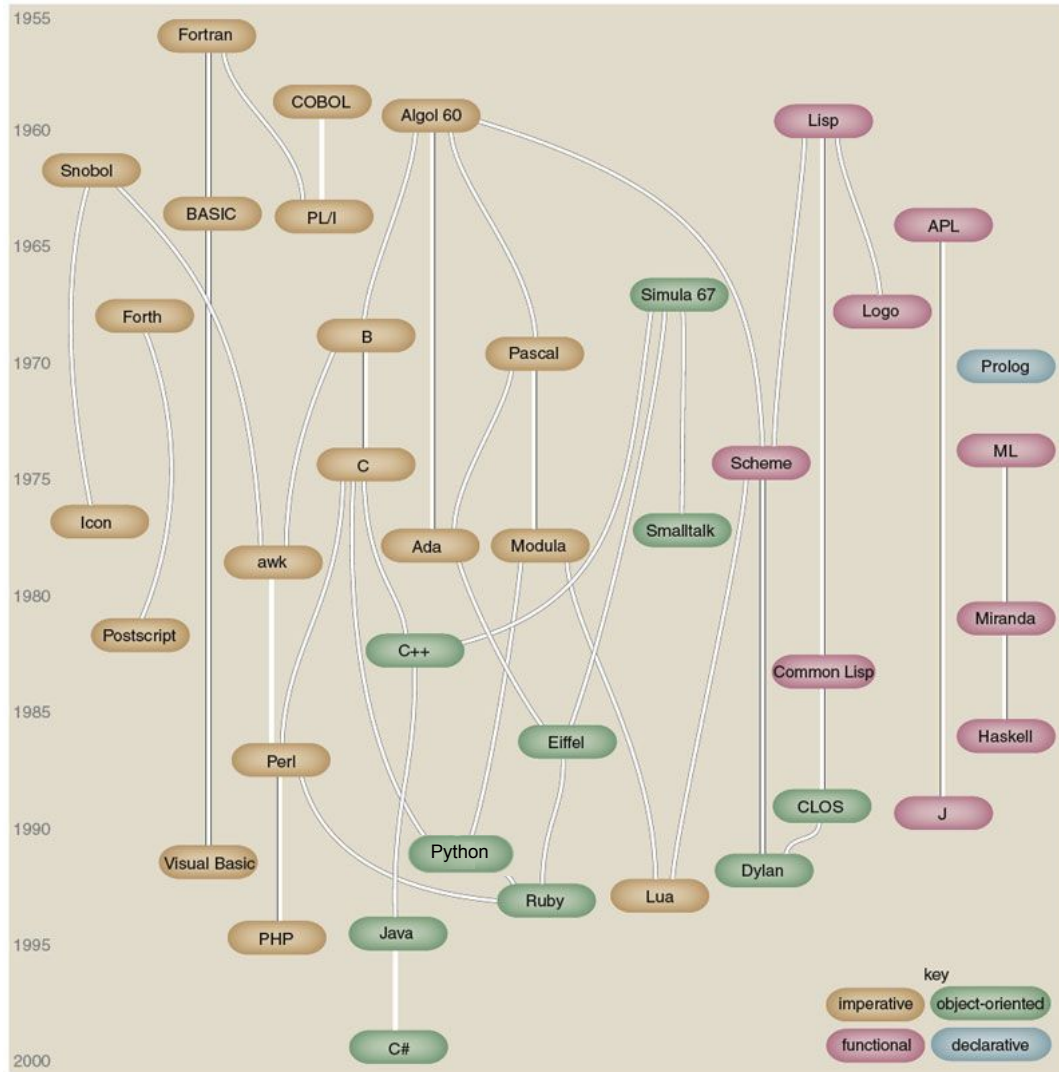
## Language Ranking: IEEE Spectrum

| Rank | Language   | Type    | Score |
|------|------------|---------|-------|
| 1    | Python     | 🌐 🖥️ ⚙️ | 100.0 |
| 2    | Java       | 🌐 📱 🖥️  | 95.3  |
| 3    | C          | 📱 🖥️ ⚙️ | 94.6  |
| 4    | C++        | 📱 🖥️ ⚙️ | 87.0  |
| 5    | JavaScript | 🌐       | 79.5  |
| 6    | R          | 🖥️      | 78.6  |
| 7    | Arduino    | ⚙️      | 73.2  |
| 8    | Go         | 🌐 🖥️    | 73.1  |
| 9    | Swift      | 📱 🖥️    | 70.5  |
| 10   | Matlab     | 🖥️      | 68.4  |

Fonte:

<https://spectrum.ieee.org/top-programming-languages/>

# Genealogia das principais linguagens de programação





# Primeiro Programa em C

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1  // primeiro programa
2
3  /* pede para incluir a biblioteca padrão
4  de entrada e saída*/
5  #include <stdio.h>
6
7  // função main inicia a execução do programa
8  int main(void) {
9
10     // imprimir na tela o texto entre aspas
11     printf("Hello World\n");
12
13     // indica que o programa terminou com sucesso
14     return 0;
15 }
```

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10     // imprimir na tela o texto entre aspas
11     printf("Hello World\n");
12
13     // indica que o programa terminou com sucesso
14     return 0;
15 }
```

Biblioteca necessária para trabalhar com comandos de entrada (scanf) e saída (printf) - stdio - standard input-output

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do
8 int main(void) {
9
10 // imprimir na tela o texto entre aspas
11 printf("Hello World\n");
12
13 // indica que o programa terminou com sucesso
14 return 0;
15 }
```

Todo programa em C precisa ter a função **main**, o programa vai começar por ela

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10     // imprimir na tela o texto entre aspas
11     printf("Hello World\n");
12
13     // indica que o programa terminou com sucesso
14     return 0;
15 }
```

**printf** - exibe na tela o que estiver entre aspas; **\n** serve como quebra de linha



## Programa Simples em C:

### Imprimindo uma linha de texto

- O caractere especial `\`
  - `\` é utilizado para modificar o significado de um caractere
  - `\n`, por exemplo, serve como *new line* ou **quebra de linha**
  - Outro exemplo relevante é o `\t` que serve para **tabulação**

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10     // imprimir na tela o texto entre
11     printf("Hello World\n");
12
13     // indica que o programa terminou
14     return 0;
15 }
```

Normalmente a última linha da função main, indica que o programa executou com sucesso; além disso, **return** sai da função main

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10 // imprimir na tela o texto entre aspas
11 printf("Hello World\n");
12
13 // indica que o programa terminou com sucesso
14 return 0;
15 }
```

No C, os blocos de código são delimitados por chaves {}



# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída */
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10     // imprimir na tela o texto entre aspas
11     printf("Hello World\n");
12
13     // indica que o programa terminou com sucesso
14     return 0;
15 }
```

**// ou /\* texto \*/ são comentários**

# Programa Simples em C:

## Imprimindo uma linha de texto

- Exemplos de comentário como cabeçalho de um programa:

```
1  /* Programa de teste
2  *  AUTOR: Danilo Perico
3  *  DATA: 05/02/2020
4  */
```

ou

```
1  /*****
2  *  Programa de teste
3  *  AUTOR: Danilo Perico
4  *  DATA: 05/02/2020
5  *****/
```

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10     // imprimir na tela o texto entre aspas
11     printf("Hello World\n");
12
13     // indica que o programa terminou com sucesso
14     return 0;
15 }
```

**#include < > serve para  
inserir bibliotecas**

# Programa Simples em C:

## Imprimindo uma linha de texto

```
1 // primeiro programa
2
3 /* pede para incluir a biblioteca padrão
4 de entrada e saída*/
5 #include <stdio.h>
6
7 // função main inicia a execução do programa
8 int main(void) {
9
10     // imprimir na tela o texto entre aspas
11     printf("Hello World\n");
12
13     // indica que o programa termina
14     return 0;
15 }
```

No C, as instruções terminam com ponto-e-vírgula ;

# Exercícios



# Exercícios

1. Escreva um programa em C que tenha a seguinte saída (comente o código):

```
C
é uma linguagem de programação
muito legal!
```

## Exercícios

2. Escreva um programa em C que tenha a seguinte saída (comente o código):

```
Produto 1          =    25.00
Produto 2          =    47.50
Produto 3          =    68.25
-----
Total              =   140.75
```

Dica: utilize o `\t`

# Variáveis e Tipos de Dados Básicos





## Variáveis

Uma variável é um nome que se refere a um dado ou valor



## Variáveis

**Uma variável é um recurso utilizado nos programas para escrever e ler dados da memória do computador!**



## Variáveis

**São simplesmente espaços na memória o qual reservamos e damos nomes**



## Tipos Básicos de Dados

- **Dados** são as entidades mais fundamentais que um programa manipula!
- Os dados podem ser de diferentes **tipos**



# Tipos Básicos de Dados

- Existem 5 tipos de dados básicos em C:
  - **char**: tipos usados para guardar caracteres, mas que também são usados para guardar números de 8 bits
  - **int**: números inteiros de 16 ou 32 bits, dependendo da implementação.
  - **float**: números reais IEEE de 32 bits.
  - **double**: números reais IEEE de 64 bits.
  - **void**: usada para criar procedimentos ou ponteiros genéricos. É o tipo nulo.



# Variáveis

- Variáveis em linguagem C:
  - Sempre que desejamos guardar um valor, que pode ser alterado durante a execução do programa, devemos utilizar **variáveis**



# Variáveis

- Variáveis em linguagem C:
  - Uma variável deve ser sempre definida antes de ser usada
  - A definição deve ser feita seguindo a sintaxe abaixo:

**tipo nome\_da\_variável;**

- Exemplos:

```
int i; //i é uma variável do tipo inteiro
char chl, novo; //chl e novo são vars do tipo char;
float pi, raio;
double total;
```



# Variáveis - Nomes

- Regras para os nomes das variáveis:
  - O **primeiro caractere** deve ser uma **letra**, e os caracteres seguintes devem ser letras, números ou sublinhado.
  - Não possuem limite de tamanho e pelo menos os 6 primeiros são significativos.
  - A linguagem C é **Case Sensitive**.
  - Exemplos:
    - Corretos: count, teste1, high\_power, \_main
    - Incorretos: 1cont, hi!there, joao&maria





# Variáveis - Nomes - Palavras Reservadas

- C possui **32 palavras reservadas**:

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



# Variáveis

- Os tipos de dados básicos podem ser modificados pelas seguintes palavras reservadas:
  - **signed:** deixa com sinal (+ ou -)
  - **unsigned:** retira o sinal (e duplica a precisão)
  - **short:** diminui a precisão na metade dos bits usados
  - **long:** dobra a precisão dos inteiros

# Variáveis



| Tipo    | Tamanho  | Valores Possíveis  |
|---------|----------|--|
| bool    | 1 byte   | true e false   |
| byte    | 1 byte   | 0 a 255  |
| sbyte   | 1 byte   | -128 a 127   |
| short   | 2 bytes  | -32768 a 32767   |
| ushort  | 2 bytes  | 0 a 65535  |
| int     | 4 bytes  | -2147483648 a 2147483647   |
| uint    | 4 bytes  | 0 to 4294967295  |
| long    | 8 bytes  | -9223372036854775808L to 9223372036854775807L                    |
| ulong   | 8 bytes  | 0 a 18446744073709551615   |
| float   | 4 bytes  | Números até 10 elevado a 38. Exemplo: 10.0f, 12.5f               |
| double  | 8 bytes  | Números até 10 elevado a 308. Exemplo: 10.0, 12.33               |
| decimal | 16 bytes | números com até 28 casas decimais. Exemplo 10.991m, 33.333m      |
| char    | 2 bytes  | Caracteres delimitados por aspas simples. Exemplo: 'a', 'ç', 'o' |

# Saída e Entrada de Dados

## *stdio.h*



**printf( )**  
**Saída na tela**



## Comando printf()

- Conforme já vimos, podemos exibir dados na tela com o comando `printf( )` da biblioteca `stdio.h`



## Comando printf()

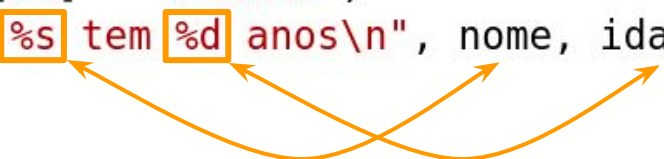
- Para a impressão de dados na tela, podemos combinar texto com o valor de alguma variável
- Exemplo:

```
7  #include <stdio.h>
8
9  int main(void) {
10     int idade = 25;
11     char nome[20] = "Fulano";
12     printf("O %s tem %d anos\n", nome, idade);
13     return 0;
14 }
```

## Comando printf()

- No exemplo temos os **especificadores de conversão**:

```
7  #include <stdio.h>
8
9  int main(void) {
10     int idade = 25;
11     char nome[20] = "Fulano";
12     printf("O %s tem %d anos\n", nome, idade);
13     return 0;
14 }
```



- Não podemos escrever **nome** e **idade** diretamente dentro da *string*
- O especificador marca o lugar que terá o valor da variável





## Comando printf()

- Os **especificadores de conversão** são diferentes para cada tipo de dado
- Alguns exemplos:

| especificador | tipo                              |
|---------------|-----------------------------------|
| %d            | Número inteiro (int)              |
| %c            | Caractere (char)                  |
| %f            | Números decimais (float e double) |
| %s            | Cadeia de caracteres (char[ ])    |



## Função printf()

- Exemplo: **Limitando** a quantidade de **casas decimais** na impressão de **números reais**:
  - Basta incluir **.x** entre o **%** e o **f**, onde **x** é o número desejado de casas decimais
  - Exemplo:

```
7  #include <stdio.h>
8
9  int main(void) {
10     double preco;
11     preco = 2.654987123;
12     printf("O preço é %.2f \n", preco);
13     return 0;
14 }
```

Saída na tela:

**O preço é 2.65**

# `scanf( )`

## Leitura de Dados



## Comando scanf()

- Para lermos valores podemos utilizar o comando **scanf( )** também da biblioteca **stdio.h**
- O **scanf()** funciona de forma semelhante ao **printf()**:
  - Também utiliza o **especificador de tipo**

## Comando scanf()

- Exemplo de leitura com o `scanf()`:

```
7  #include <stdio.h>
8
9  int main(void) {
10     float preco;
11     scanf("%f ", &preco);
12     printf("O preço é %.2f \n", preco);
13     return 0;
14 }
```

- Para ler variáveis dos tipos básicos, é preciso preceder o nome de cada variável de um `&` (“E” comercial)

# Exercícios



## Exercícios

3. Faça um programa em C que solicita do usuário dois números reais, então some os dois números e exiba o resultado.
4. Faça um programa que solicita do usuário uma quantidade de dias, horas, minutos e segundos. Calcule o total convertido em somente segundos e mostre o resultado.
5. Faça um programa que solicita dois números inteiros ao usuário e armazene cada um destes números em variáveis distintas. Realize então uma troca de valores entre as variáveis. Exiba, no fim, os valores de cada variável. Exemplo:

Entrada:  $a = 9$ ;  $b = 2$ ;      Saída:  $b = 2$ ;  $a = 9$



## Exercícios

6. Escreva um programa que pergunte a quantidade de km percorridos por um carro alugado, assim como a quantidade de dias pelos quais o carro foi alugado. Calcule e mostre o preço a pagar, sabendo que o carro custa R\$ 60,00 por dia e R\$ 0,15 por km rodado.
7. Tendo como dados de entrada a altura de uma pessoa, construa um algoritmo que calcule seu peso ideal, usando a seguinte fórmula:  $(72.7 * \text{altura}) - 58$ . Imprima o resultado.