

# Sintaxe Básica e Expressões

## PE-03 – v1.2

**Prof. Paulo Joia Filho**

- 1 Introdução
- 2 Sintaxe Básica
- 3 Expressões
- 4 Conclusões

# 1 Introdução

- Objetivos

## 2 Sintaxe Básica

## 3 Expressões

## 4 Conclusões

# Objetivos

Os principais objetivos desta aula são:

- Explorar os comandos básicos da linguagem C.
- Praticar através dos exemplos e exercícios em sala.

## 1 Introdução

## 2 Sintaxe Básica

- Variáveis e tipos de dados
- Comandos de entrada e saída

## 3 Expressões

## 4 Conclusões

# Começando...

[hello.c]

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Tinha que ser hello world!\n");
5     return 0;
6 }
```

No Linux, salve o programa acima como hello.c, compile com:

```
$ gcc hello.c -o hello
```

E execute com:

```
$ ./hello
```

# Declaração de variáveis

- Para usar uma variável em C, ela deve ser definida indicando o seu tipo e nome:

```
tipo nome1 [, nome2]... ;
```

- Pode-se definir mais de uma variável em uma linha, bastando separá-las por vírgula. Ex:

```
float valor_salario;  
char sexo;  
int i,k,j;
```

- A linguagem C permite inicializar uma variável quando ela é declarada.
  - O sinal de igual “=” é o operador de atribuição de valores;
  - Ela deve receber um valor compatível com o seu tipo.

```
float valor_salario = 15000;  
char sexo = 'M';
```

# Declaração de variáveis

- Variáveis podem ser declaradas:
  - Dentro de funções (variáveis locais)
  - Como parâmetros de funções (parâmetros formais)
  - Fora de todas as funções (variáveis globais)
- Modificador **const** – impede que o valor da variável seja modificado pelo programa.

Ex:

```
const float valor_salario = 5000; // nunca receberá aumento  
const char sexo = 'M';
```



# Inicialização de variáveis

## Importante!

É ideal que todas as variáveis sejam inicializadas antes de serem utilizadas.

Quando uma variável é declarada, o programa reserva um espaço de memória para ela:

- Mas o espaço alocado não é inicializado.
- Portanto, uma variável não inicializada pode conter qualquer valor!

## Nomes de variáveis

- Toda variável deve ter um nome único dentro de um mesmo contexto.
- Para nomear uma variável é necessário seguir algumas regras impostas pelo compilador:
  1. O nome de uma variável deve começar por uma letra ou por um carácter “\_” (*underline*).
  2. Os demais caracteres de uma variável podem ser letras, dígitos ou “\_”.
  3. O compilador C diferencia letras maiúsculas de minúsculas.

C é case-sensitive!

# Legibilidade do código

## Organização

*O processo de escolha de nomes de variáveis é importante para a legibilidade do programa e manutenções posteriores.*

Alguns guias para nos auxiliar nesta tarefa:

- Estilo de código

<https://users.ece.cmu.edu/~eno/coding/CCodingStandard.html>

- Indentação K & R

<https://gist.github.com/jesseschalken/0f47a2b5a738ced9c845>

# Legibilidade do código

## Comentários

- Comentários ajudam a aumentar a clareza do código.
- Os comentários de um programa devem ser colocados entre `/*` e `*/`.
- Quaisquer textos colocados entre estes dois símbolos serão ignorados pelo compilador.
- A partir da padronização C99, o compilador aceita também comentários com `//`.
- A diferença básica entre eles é que o `//` comenta apenas uma linha.

Ex:

```
/*  
    Este é um comentário  
    de 02 linhas  
*/  
int i; // índice do vetor de saída  
float soma; /* Soma dos valores pagos */  
...
```

# Tipos básicos

Para criar variáveis em C deve-se indicar para o compilador qual o tipo de dado desta variável.

A linguagem C define os seguintes tipos básicos de dados:

- Caractere (**char**)
- Inteiro (**int**)
- Ponto flutuante (**float**)
- Ponto flutuante de precisão dupla (**double**)

Modificadores de tipo:

- **signed**
- **unsigned**
- **short**
- **long**

**Nota:** o tipo **void** não é utilizado para variáveis, mas sim para indicar que uma função não retorna valores ou não possui parâmetros de entrada.

# Limites × Tamanho ocupado na memória

Tipo	Número de Bits	Formato para leitura e impressão	Início	Fim
_Bool	8	Não tem (pode-se utilizar %d)	0	1
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
int	32	%i	-2.147.483.648	2.147.483.647
unsigned int*	32	%u	0	4.294.967.295
signed int	32	%i	-2.147.483.648	2.147.483.647
long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
signed long int	32	%li	-2.147.483.648	2.147.483.647
float	32	%f	3,4E-38	3,4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	128	%Lf	3,4E-4932	3,4E+4932

**Figura:** Valores para arquitetura de 32 bits (-with-arch-32=i686)

## Sobre o tipo `_Bool`

- O tipo `_Bool` foi criado na versão C99, ele não existia na versão C89/90.
- Detalhe: se utilizarmos o cabeçalho `#include <stdbool.h>`, ele fornece `bool` como uma macro para `_Bool`.
- O foco deste curso é a versão C99, portanto, ele pode ser utilizado nos exercícios, embora não seja estritamente necessário;
  - ✎ Ele pode ser perfeitamente substituído por um `char` que ocupa a mesma quantidade de memória, 8 bits.

Lembrando que, na linguagem C, em comparações lógicas, 0 (zero) é considerado falso e diferente de 0 (zero) é considerado verdadeiro.

# Padrões da linguagem C

Na Série 5 do GCC, o modo padrão para C é `-std=gnu11` ao invés de `-std=gnu89`, confira: <https://www.gnu.org/software/gcc/gcc-5/changes.html>.

A versão C de 2011 foi aceita como padrão para o GNU C, isto significa que, quando você compila um programa com gcc sem especificar um padrão, ele assume o c11.



# Forçando um padrão de compilação

[bool.c]

```
1 #include <stdio.h>
2
3 int main() {
4     _Bool b = 1;
5
6     if (b) {
7         printf("Booleanos são aceitos em C99 e C11.\n"
8             "Um _Bool tem tamanho de %lu bits.\n\n", sizeof(b) * 8);
9     }
10    return 0;
11 }
```

Salve o programa acima como bool.c e tente compilar com **C11**:

```
$ gcc -std=c11 -pedantic bool.c -o bool
```

Com **C99**:

```
$ gcc -std=c99 -pedantic bool.c -o bool
```

E, finalmente, com **C89**:

```
$ gcc -std=c89 -pedantic bool.c -o bool
```

# Forçando um padrão de compilação

- Para **C11** e **C99** o programa anterior compila, mas para **C89** não:

```
bool.c: In function 'main':  
bool.c:4:2: warning: ISO C90 does not support boolean types [-Wpedantic]  
  Bool b = 1;  
  ^
```

- A função **sizeof** retorna o tamanho em bytes que um tipo de dado ocupa na memória. Pode-se também passar o tipo como parâmetro para a função.
- O retorno de **sizeof** é do tipo **size\_t**
  - size\_t** foi criado a partir da versão C99 por questões de portabilidade.
  - Ele assegura o maior inteiro possível para uma dada arquitetura (normalmente um `unsigned long`).
  - Está definido em **stddef.h**.

```
1  #include <stdio.h>

3  int main() {
4      printf("\n    Tipo de Dado    Tamanho em bits \n");
5      printf("===== \n");

7      printf("        _Bool: %8ld\n\n", sizeof(_Bool)*8);

9      printf("        char: %8ld\n", sizeof(char)*8);
10     printf("    unsigned char: %8ld\n", sizeof(unsigned char)*8);
11     printf("        signed char: %8ld\n\n", sizeof(signed char)*8);

13     printf("        short int: %8ld\n", sizeof(short int)*8);
14     printf("    unsigned short int: %8ld\n", sizeof(unsigned short int)*8);
15     printf("        signed short int: %8ld\n\n", sizeof(signed short int)*8);

17     printf("        int: %8ld\n", sizeof(int)*8);
18     printf("    unsigned int: %8ld\n", sizeof(unsigned int)*8);
19     printf("        signed int: %8ld\n\n", sizeof(signed int)*8);

21     printf("        long int: %8ld\n", sizeof(long int)*8);
22     printf("    unsigned long int: %8ld\n", sizeof(unsigned long int)*8);
23     printf("        signed long int: %8ld\n\n", sizeof(signed long int)*8);

25     printf("        size_t: %8ld\n\n", sizeof(size_t)*8);

27     printf("        float: %8ld\n", sizeof(float)*8);
28     printf("        double: %8ld\n", sizeof(double)*8);
29     printf("    long double: %8ld\n\n", sizeof(long double)*8);

31     return 0;
}
```

```

1  #include <stdio.h>

3  int main() {
    printf("\n    Tipo de Dado    Tamanho em bits \n");
5     printf("===== \n");

7     printf("        _Bool: %8ld\n\n", sizeof(_Bool)*8);

9     printf("        char: %8ld\n", sizeof(char)*8);
    printf("    unsigned char: %8ld\n", sizeof(unsigned char)*8);
11    printf("        signed char: %8ld\n\n", sizeof(signed char)*8);

13    printf("        short int: %8ld\n", sizeof(short int)*8);
    printf("    unsigned short int: %8ld\n", sizeof(unsigned short int)*8);
15    printf("        signed short int: %8ld\n\n", sizeof(signed short int)*8);

17    printf("        int: %8ld\n", sizeof(int)*8);
    printf("    unsigned int: %8ld\n", sizeof(unsigned int)*8);
19    printf("        signed int: %8ld\n\n", sizeof(signed int)*8);

21    printf("        long int: %8ld\n", sizeof(long int)*8);
    printf("    unsigned long int: %8ld\n", sizeof(unsigned long int)*8);
23    printf("        signed long int: %8ld\n\n", sizeof(signed long int)*8);

25    printf("        size_t: %8ld\n\n", sizeof(size_t)*8);

27    printf("        float: %8ld\n", sizeof(float)*8);
    printf("        double: %8ld\n", sizeof(double)*8);
29    printf("    long double: %8ld\n\n", sizeof(long double)*8);

31    return 0;
    }

```

Tipo de Dado	Tamanho em bits
-----	-----
_Bool:	8
char:	8
unsigned char:	8
signed char:	8
short int:	16
unsigned short int:	16
signed short int:	16
int:	32
unsigned int:	32
signed int:	32
long int:	64
unsigned long int:	64
signed long int:	64
size_t:	64
float:	32
double:	64
long double:	128

# Conversão de tipos

## Typecasting

- De uma forma geral, pode-se realizar a conversão de um tipo para outro na linguagem C (técnica conhecida como *typecasting*).
- Uso:
  - Melhorar o entendimento de alguns trechos do programa;
  - Compatibilizar variáveis;
  - Compatibilizar um determinado tipo de parâmetro na chamada de uma função para o tipo de parâmetro esperado.
- A sintaxe do “typecasting” é a seguinte:

(tipo) variável

Ex:

```
int i = 10;  
float n = (float) i;
```

# Conversão de tipos

[cast.c]

## Exemplo

```
1  #include <stdio.h>

3  int main() {
    float r1 = 7 / 2;
5   float r2 = ((float) 7) / 2;
    float r3 = 7 / ((float) 2);
7   float r4 = ((float) 7) / ((float) 2);
    float r5 = (float) 7 / 2; // typecast tem precedência
9   float r6 = (float) (7 / 2); // sobre o operador de divisão.

11  printf("\n Saída:\n");
    printf("\n %.1f %.1f %.1f", r1, r2, r3);
13  printf("\n %.1f %.1f %.1f\n", r4, r5, r6);

15  return 0;
}
```

Qual a saída deste programa?

# Conversão de tipos

[cast.c]

## Exemplo

```
1  #include <stdio.h>

3  int main() {
    float r1 = 7 / 2;
5   float r2 = ((float) 7) / 2;
    float r3 = 7 / ((float) 2);
7   float r4 = ((float) 7) / ((float) 2);
    float r5 = (float) 7 / 2; // typecast tem precedência
9   float r6 = (float) (7 / 2); // sobre o operador de divisão.

11  printf("\n Saída:\n");
    printf("\n %.1f %.1f %.1f", r1, r2, r3);
13  printf("\n %.1f %.1f %.1f\n", r4, r5, r6);

15  return 0;
}
```

Qual a saída deste programa?

Saída:

3.0 3.5 3.5  
3.5 3.5 3.0

# Caracteres para formatação de entrada e saída

Código	Formato
<b>\n</b>	nova linha
<b>\t</b>	tab
<b>\b</b>	retrocesso
<b>\"</b>	aspas
<b>\\</b>	barra
<b>\f</b>	salta formulário
<b>\0</b>	nulo

Código	Formato
<b>%c</b>	caractere
<b>%d</b>	inteiro
<b>%i</b>	inteiro
<b>%e</b>	notação científica
<b>%E</b>	notação científica
<b>%f</b>	ponto flutuante
<b>%g</b>	utiliza %e ou %f, o que for mais curto
<b>%G</b>	utiliza %E ou %F, o que for mais curto
<b>%o</b>	octal
<b>%s</b>	cadeia de caracteres
<b>%u</b>	inteiro sem sinal
<b>%x</b>	hexadecimal (letras minúsculas)
<b>%X</b>	hexadecimal (letras maiúsculas)
<b>%p</b>	ponteiro
<b>%%</b>	imprime o sinal %



# Saída de dados

`[saida1.c]`

A biblioteca `stdio.h` é utilizada para entrada e saída de dados.

*Sintaxe:*

```
printf("expressão de controle", argumentos);
```

*Exemplo:*

```
1  #include <stdio.h>
2
3  int main() {
4      printf(" \n Este é o comando de saída de dados.");
5      printf(" \n Este é o número dois: %d.", 2);
6      printf(" \n %s está a %d milhões de milhas \n do sol.", "Vênus", 67);
7      printf(" \n A letra %c ", 'j');
8      printf(" \n Pronuncia—se %s.\n\n", "jota.");
9
10     return 0;
11 }
```

# Saída de dados

[saida1.c]

A biblioteca `stdio.h` é utilizada para entrada e saída de dados.

*Sintaxe:*

```
printf("expressão de controle", argumentos);
```

*Exemplo:*

```
1 #include <stdio.h>
2
3 int main() {
4     printf(" \n Este é o comando de saída de dados.");
5     printf(" \n Este é o número dois: %d.", 2);
6     printf(" \n %s está a %d milhões de milhas \n do sol.", "Vênus", 67);
7     printf(" \n A letra %c ", 'j');
8     printf(" \n Pronuncia-se %s.\n\n", "jota.");
9
10    return 0;
11 }
```

```
Este é o comando de saída de dados.
Este é o número dois: 2.
Vênus está a 67 milhões de milhas
do sol.
A letra j
Pronuncia-se jota..
```

# Saída de dados

[saida2.c]

## Exemplo

```
1  #include <stdio.h>

3  int main() {
    // tamanho de campos na impressão
5  printf("\n %2d" , 350);

7  // arredondamento
    printf("\n %4.2f" , 3456.78);

9  // alinhamento
11 printf("\n %10.2f %10.2f %10.2f", 8.0, 15.3, 584.13);

13 // complemento de zeros a esquerda
    printf("\n %04d" , 21);

15 // impressão de caracteres
17 printf("%d %c %x %o \n\n" , 'A', 'A', 'A', 'A');

19 return 0;
}
```

# Saída de dados

[saida2.c]

## Exemplo

```
1  #include <stdio.h>

3  int main() {
    // tamanho de campos na impressão
5  printf("\n %2d" , 350);

7  // arredondamento
    printf("\n %4.2f" , 3456.78);

9  // alinhamento
11 printf("\n %10.2f %10.2f %10.2f", 8.0, 15.3, 584.13);

13 // complemento de zeros a esquerda
    printf("\n %04d" , 21);

15 // impressão de caracteres
17 printf("%d %c %x %o \n\n" , 'A', 'A', 'A', 'A');

19 return 0;
}
```

```
350
3456.78
      8.00      15.30      584.13
002165 A 41 101
```

# Entrada de dados

- Utilizar a biblioteca `stdio.h` para entrada e saída de dados.
  - `scanf("expressão de controle", argumentos);`
  - `scanf("%_", &);`
- Argumentos: devem conter os endereços das variáveis. A linguagem C oferece um operador para tipos básicos chamado **operador de endereço** e referenciado pelo símbolo **&** que retorna o endereço do operando.

*Exemplo:*

```
int num;  
  
scanf("%d", &num);
```

# Entrada de dados

[idade.c]

## Exemplo

```
1  #include <stdio.h>

3  int main() {
    float anos = 0, dias = 0;

5
    printf("\n Digite sua idade em anos: ");
7    scanf("%f", &anos);
    dias = anos * 365;
9    printf(" Sua idade em dias é: %.0f dias\n\n", dias);

11   return 0;
}
```

# Entrada de dados

`[idade.c]`

## Exemplo

```
1  #include <stdio.h>

3  int main() {
    float anos = 0, dias = 0;

5     printf("\n Digite sua idade em anos: ");
7     scanf("%f", &anos);
    dias = anos * 365;
9     printf(" Sua idade em dias é: %.0f dias\n\n", dias);

11    return 0;
}
```

Saída:

```
Digite sua idade em anos: 21
Sua idade em dias é: 7665 dias
```

- 1 Introdução
- 2 Sintaxe Básica
- 3 Expressões**
  - Expressões
- 4 Conclusões



# Principais Operadores

## Alguns operadores utilizados na linguagem C

Aritméticos	
Símbolo	Operação
+	adição
-	subtração
*	multiplicação
/	divisão
%	resto da divisão
++	incremento
--	decremento

Relacionais	
Símbolo	Significado
<	menor que
>	maior que
<=	menor ou igual à
>=	maior ou igual à
==	igual
!=	diferente

Lógicos	
Símbolo	Operação
&&	AND
	OR
!	NOT

# Operadores incrementais

`[incremento.c]`

## Exemplo

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0;
5      printf("\n");
6      printf(" %d\n", i++);
7      printf(" %d\n\n", i--);
8
9      printf(" %d\n", --i);
10     printf(" %d\n\n", ++i);
11
12     printf(" %d\n\n", i);
13
14     i = 6;
15     printf(" %f\n\n", ++i / 2.0);
16
17     return 0;
18 }
```

Qual a saída de cada `printf`?

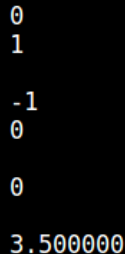


# Operadores incrementais

[incremento.c]

## Exemplo

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0;
5      printf("\n");
6      printf(" %d\n", i++);
7      printf(" %d\n\n", i--);
8
9      printf(" %d\n", --i);
10     printf(" %d\n\n", ++i);
11
12     printf(" %d\n\n", i);
13
14     i = 6;
15     printf(" %f\n\n", ++i / 2.0);
16
17     return 0;
18 }
```



```
0
1
-1
0

0

3.500000
```

Qual a saída de cada `printf`?

# Operador unário “-”

- É usado para realizar troca de sinal algébrico do valor.
- Pode-se também multiplicar o valor do operando por -1.

*Exemplos:*

```
num = -10;
```

```
num = -num;
```

## Operador ternário “? :”

**variável = <condição> ? se\_verdadeiro : se\_falso;**

Ex:

```
y = x > 9 ? 100 : 200;
```

- Se  $x = 10$ ,  $y$  recebe o valor 100;
- Se  $x \leq 9$ ,  $y$  recebe o valor 200.
- Pode ser usado como entrada para outros comandos.

Ex:

```
printf("O maior numero é: %i \n", a > b ? a : b);
```

## 1 Introdução

## 2 Sintaxe Básica

## 3 Expressões

## 4 Conclusões

- Exercícios de aprendizagem
- Referências bibliográficas

## Exercícios de Aprendizagem

### Exercício 1

Usando **sizeof** faça um programa para exibir os tamanhos dos tipos básicos **char**, **int**, **float**, **double**. Use **printf** para formatar a saída.

*Dica:* siga o modelo do programa `sizeof.c` apresentado anteriormente.

### Exercício 2

Escreva um programa para encontrar o volume de uma esfera a partir do raio  $R$ , fornecido pelo usuário. O volume da esfera pode ser calculado como:

$$V = \frac{4}{3}\pi R^3$$

- Siga o modelo do programa apresentado para calcular a área do círculo, porém, com as seguintes melhorias:
  - Valide o valor de entrada do raio,  $R > 0$ ;
  - Desta vez, use a constante **M\_PI** da biblioteca **math**.

# Exercícios de Aprendizagem

## Exercício 3

Faça um programa para calcular o logaritmo dos números abaixo na base natural e na base 10. Use a função `printf` para exibir os resultados com apenas 3 casas decimais.

a) 3141

b) 32,37

### Dicas:

- Use as funções `log` e `log10` da biblioteca `math`;
- Valide os resultados usando Python no console do Linux. Como?



# Exercícios de Aprendizagem

## Mudança de base do logaritmo

A expressão abaixo mostra como se efetua a mudança de um logaritmo de base  $b$  para um logaritmo de base  $c$  (arbitrária):

$$\log_b a = \frac{\log_c a}{\log_c b},$$

onde:  $a > 0$ ;  $0 < b \neq 1$ ;  $0 < c \neq 1$ .

## Exercício 4

**Logaritmo em qualquer base.** Com base na definição acima, faça um programa em C para calcular o logaritmo em qualquer base.

- Entrada: logaritmando ( $a$ ), base ( $b$ );
- Saída:  $x = \log_b a$ ;
- Condições: garantir que  $a > 0$  e  $0 < b \neq 1$ ;

# Exercícios de Aprendizagem

## *Exercício 4: exemplos de saída*

```
*** Logaritmo em qualquer base ***
```

```
Entre logaritmando e base: 4096 8  
Log de 4096 na base 8 é 4
```

```
*** Logaritmo em qualquer base ***
```

```
Entre logaritmando e base: 5 1  
Entrada inválida, verifique as condições:  
-- logaritmando > 0;  
-- base > 0 e base diferente de 1.
```

# Exercícios de Aprendizagem

## Exercício 5

**Valores Aleatórios.** Faça um programa que sorteie dois números inteiros entre 1 e 100, em seguida:

- exiba os dois números;
- exiba o maior deles usando o operado ternário.

### Dicas:

- Use as funções **srand** e **rand** do cabeçalho **stdlib.h** para fazer o sorteio;
- Use a função **time** do cabeçalho **time.h** para definir a semente.

## Exercícios de Aprendizagem

### Exercício 6

**Escala Termométrica.** Faça um programa que converta a temperatura da escala Celsius para outras duas escalas de temperatura, Fahrenheit e Kelvin.

Dado de entrada: temperatura Celsius ( $t_C$ )

Dados de saída: temperatura Fahrenheit ( $t_F$ ) e Kelvin ( $t_K$ )

Onde:

$$t_F = 32 + \frac{9t_C}{5} \qquad t_K = t_C + 273,15$$

Obs: crie duas funções de conversão para praticar,  $t_C \rightarrow t_F$  e  $t_C \rightarrow t_K$ .

### *Exercício 6: exemplo de saída*

```
Informe a temperatura Celsius: 25
Fahrenheit = 77.00
Kelvin = 298.15
```

# Exercícios de Aprendizagem

## Exercício 7

**Cálculo Estrutural.** Faça um algoritmo que retorne o máximo momento fletor, **Mmax**, e máxima flecha,  **$\delta_{\max}$** , numa viga biapoiada, de vão **L**, com carregamento distribuído uniforme retangular, **q**, sabendo que:

$$M_{\max} = \frac{qL^2}{8} \qquad \delta_{\max} = \frac{5qL^2}{384EI}$$

Onde:

E – módulo de elasticidade do material da viga

I – momento de inércia da seção transversal da viga

Teste com os valores:

$$\begin{array}{ll} q = 12 \text{ kNm} & E = 29 \times 10^6 \text{ kN/m}^2 \\ L = 5 \text{ m} & I = 0,003 \text{ m}^4 \end{array}$$

# Exercícios de Aprendizagem

## *Exercício 7: exemplo de saída*

```
Informe um valor para q: 12  
Informe um valor para L: 5  
Informe um valor para E: 29e6  
Informe um valor para I: 0.003
```

```
Mmax = 37.5 kN m^3  
Dmax = 4.48994e-05 m
```

# Referências Bibliográficas I



Aguilar, L. J. (2008).

*Programação em C++: Algoritmos, Estruturas de Dados e Objetos.*  
McGraw-Hill, São Paulo.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2002).

*Algoritmos: Teoria e Prática.*  
Elsevier, Rio de Janeiro.



Drozdek, A. (2009).

*Estrutura de Dados e Algoritmos em C++.*  
Cengage Learning, São Paulo.



Forbellone, A. L. V. e Eberspacher, H. F. (2005).

*Lógica de Programação: A Construção de Algoritmos e Estrutura de Dados.*  
Pearson Prentice Hall, São Paulo, 3 edition.



Gatto, E. C. (2013).

Introdução à Linguagem C - Notas de Aula.  
Universidade Sagrado Coração (USC).  
Disponível em: <<https://pt.slideshare.net/elainececiliagatto/pc-introduo>>. Acesso em:  
2018-09-16.

# Referências Bibliográficas II



Knuth, D. E. (2005).

*The Art of Computer Programming.*

Addison-Wesley, Upper Saddle River, NJ, USA.



Pinheiro, F. d. A. C. (2012).

*Elementos de Programação em C.*

Bookman, Porto Alegre.



Sedgewick, R. (1998).

*Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, Searching.*

Addison-Wesley, Boston, 3rd edition.



Szwarcfiter, J. L. e Markenzon, L. (1994).

*Estruturas de Dados e Seus Algoritmos.*

LTC, Rio de Janeiro.



Tenenbaum, A. A., Langsam, Y., e Augenstein, M. J. (1995).

*Estruturas de Dados Usando C.*

Makron Books, São Paulo.