

Algoritmos em linguagem C

Alexsandro Santos Soares
`prof.asoares@gmail.com`

Universidade Federal de Uberlândia
Faculdade de Computação

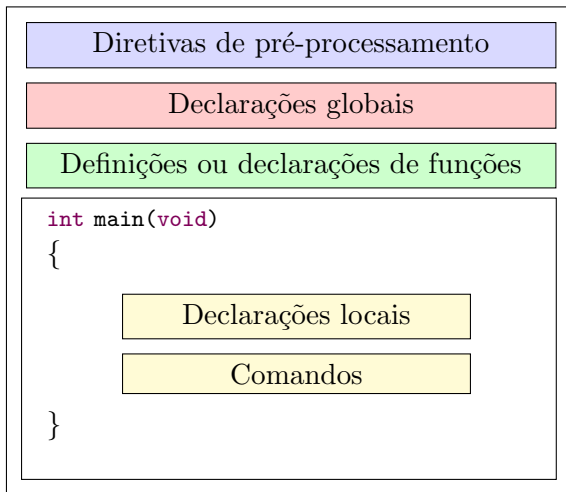
Sumário

- 1 Introdução
- 2 Programas em C
 - Comentando programas
 - Alô mundo
 - Divisão inteira
 - Soma de inteiros
 - Paridade
 - Equações do segundo grau
 - Números complexos
 - Triângulos
 - Peso ideal
 - Fatorial
 - Progressão aritmética
- 3 Para saber mais
- 4 Fontes

Introdução

- Antes de continuarmos com o estudo dos algoritmos, faremos uma breve pausa para introduzir os conceitos necessários para programá-los em linguagem C.
- Neste curso usaremos a última versão padronizada da linguagem:
 - C11, um nome informal para ISO/IEC 9899:2011.

Estrutura de um programa em C



Um programa em C

```
#include <stdio.h>

int main(void)
{
    printf("Alô mundo\n");
    return 0;
}
```

Um programa em C

Diretiva de pré-processamento para incluir funções de entrada e saída no programa

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

```
int main(void)
{
    printf("Alô mundo\n");
    return 0;
}
```

Um programa em C

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

```
int main(void)
```

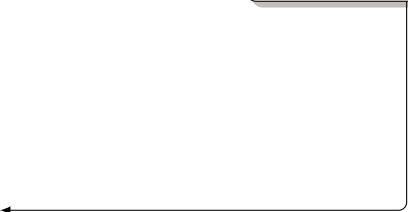
```
{
```

```
    printf("Alô mundo\n");
```

```
    return 0;
```

```
}
```

Escreve Alô mundo na tela, salta uma linha e termina o programa

A diagram consisting of a vertical line extending downwards from the callout box, followed by a horizontal line that points left towards the `printf` statement in the code block.

Comentando programas

- Algumas vezes o significado de uma seção de código não é inteiramente claro.
 - Isso é especialmente verdadeiro em C.
- O programador que escreveu o programa deve colocar alguns **comentários** no código para auxiliar seu futuro leitor.
- Esses comentários são chamados de **documentação interna do programa**.
- Esses comentários são descartados pelo compilador ao gerar o programa executável.
- C usa dois tipos de comentários: comentários de bloco e comentários de linha.

Comentário de bloco

- Um **comentário de bloco** é usado quando o comentário precisar de muitas linhas para ser expressado.
- Para abrir este tipo de comentário usamos `/*` e para fechar, `*/`.
- Quaisquer caracteres entre a abertura e o fechamento do comentário será descartado pelo compilador.
- Abaixo está um exemplo com dois comentários de bloco

```
1 /* Este é um comentário de bloco que
2    envolve duas linhas.                */
3
4 /*
5  * É um estilo muito comum colocar a abertura em uma linha
6  * isolada, seguida pela documentação e depois pelo fechamento
7  * em outra linha isolada. Programadores gostam de colocar
8  * asteriscos no início de cada linha para demarcar claramente
9  * o comentário.
10 */
```

Comentário de linha

- Um **comentário de linha** usa duas barras (//) para definir um comentário.
- Esse comentário termina quando a linha termina. Assim, não há a necessidade de fechamento.
- É normalmente usado para comentários curtos e pode iniciar em qualquer lugar na linha.
- Abaixo está um exemplo com dois comentários de linha.

```
1 // Este é um comentário que envolve a linha inteira
2
3 a = 5;    // Este é um comentário de linha parcial
```

Padronização da codificação

- Infelizmente muitos programadores documentam seus códigos-fonte, ou parte deles, somente após o *término* da implementação:
 - Dificulta a manutenibilidade e a reutilização do código.
 - Código e documentação ficam fora de sincronia.
- A documentação externa, distante do código-fonte, confunde a equipe futuras de desenvolvimento com informações desatualizadas.
- A utilização de um padrão de codificação é importante para o desenvolvimento **colaborativo** de software.
 - Uma formatação e denominação padrão de código facilita a compreensão.
 - O compartilhamento deste padrão pela equipe de desenvolvimento permite que um membro compreenda rapidamente o código escrito por outros.
 - A consequência pode ser uma melhoria significativa na qualidade do produto final.

Padronização de comentários

- Existem vários padrões de codificação que podem ser adotados para desenvolvimento em C ou em outras linguagens de programação.
- Aqui adotaremos uma versão simplificada para padronizar a documentação interna visando criar o hábito de documentar o código.
- Existem várias ferramentas que podem extrair a documentação do código e transformá-lo em vários outros formatos: doc, rft, XML, L^AT_EX, HTML, etc.
- Uma dessas ferramentas é o Doxygen. É dela que adotaremos o formato de comentários.

O primeiro exemplo reescrito

```
1  /**
2   * @file exemplo1.c
3   * @brief Demonstra alguns dos componentes de um programa C simples.
4   *       Esse exemplo é bastante básico.
5   *
6   * Neste ponto vai uma explicação mais detalhada do que este módulo
7   * se propõe e pode envolver várias linhas.
8   *
9   * @author Aldovandro Cantagalo (Zé Lindinho)
10  * @author Macunaíma
11  * @date 21/03/2018
12  * @bugs Nenhum conhecido.
13  */
14  #include <stdio.h>
15
16  int main(void)
17  {
18      printf("Alô mundo\n");
19      return 0;
20  } // main
```

O GNU Compiler Collection (GCC)

- O GNU Compiler Collection (GCC) pode compilar programas escritos em C, C++, Objective-C, Fortran, Ada e Go.
- O GCC é um ótimo compilador de código livre e será usado na disciplina.
 - Existe uma outra opção igualmente boa: o clang para o LLVM.
- No laboratório ele já está instalado.
- Em casa você deve instalar, talvez com o auxílio de outros colegas mais experientes tais como o monitor da disciplina.
- Dependendo do compilador usado pode haver diferenças nas chamadas.
 - Consulte a documentação apropriada.

Compilando e executando o programa

Para compilar um arquivo de nome `exemplo1.c`, digite na linha de comando:

```
> gcc -std=c11 -c exemplo1.c
```

Se o seu código estiver correto, o gcc criará um arquivo objeto binário com extensão `.o`:

```
> ls  
exemplo1.c  exemplo1.o
```

Para gerar o arquivo executável final, digite

```
> gcc exemplo1.o -o exemplo1.exe
```

Para executar este arquivo:

```
> ./exemplo1.exe  
Alô mundo
```

Note que `>` é o marcador da linha de comando e não faz parte das chamadas.

O algoritmo da divisão inteira em C

Considere novamente o algoritmo da divisão inteira:

```
1 leia o número natural  $A$ 
2 leia o número natural  $B$ 
3  $C \leftarrow 0$ 
4 enquanto  $A \geq B$  faça
5   |    $A \leftarrow A - B$ 
6   |    $C \leftarrow C + 1$ 
7 fim enqto
8 escreva  $C$ 
9 escreva  $A$ 
```

No próximo slide iremos *implementar* este algoritmo em C.

Exemplo 2 – Divisão inteira

```
1  leia o número natural  $A$ 
2  leia o número natural  $B$ 
3   $C \leftarrow 0$ 
4
5  enquanto  $A \geq B$  faça
6      |    $A \leftarrow A - B$ 
7      |    $C \leftarrow C + 1$ 
8  fim enqto
9
10 escreva  $C$ 
11 escreva  $A$ 
```

```
12 #include <stdio.h>
13
14 int main(void)
15 {
16     // Declarações locais
17     unsigned int a = 0;
18     unsigned int b = 0;
19     unsigned int c = 0;
20
21     scanf("%u", &a);
22     scanf("%u", &b);
23     c = 0;
24
25     while (a >= b) {
26         a = a - b;
27         c = c + 1;
28     } // while
29
30     printf("%u\n", c);
31     printf("%u\n", a);
32     return 0;
33 } // main
```

O arquivo completo

```
1  /**
2   * @file exemplo2.c
3   * @brief 0 algoritmo da divisão inteira.
4   *
5   * 0 programa lerá dois números inteiros, a e b, do teclado e
6   * imprimirá o quociente e o resto inteiros de a dividido por b.
7   *
8   * @author Alexsandro Santos Soares
9   * @date 22/04/2018
10  * @bugs Se b for zero, haverá um laço infinito
11  */
12 #include <stdio.h>
13
14 int main(void)
15 {
16     // Declarações locais
17     unsigned int a = 0;
18     unsigned int b = 0;
19     unsigned int c = 0;
20
21     scanf("%u", &a);
22     scanf("%u", &b);
23     c = 0;
24
25     while (a >= b) {
26         a = a - b;
27         c = c + 1;
28     } // while
29
30     printf("%u\n", c);
31     printf("%u\n", a);
32     return 0;
33 } // main
```

Compilando e executando o programa

Para compilar e gera o executável:

```
> gcc -std=c11 -c exemplo2.c  
> gcc exemplo2.o -o exemplo2.exe
```

Para executar este arquivo com $a = 35$, $b = 6$ e saída $c = 5$, $a = 5$.

```
> ./exemplo2.exe  
35  
6  
5  
5
```

Para executar este arquivo com $a = 0$, $b = 6$ e saída $c = 0$, $a = 0$.

```
> ./exemplo2.exe  
0  
6  
0  
0
```

Testando o programa

Para executar este arquivo com $a = 35, b = 0$ e saída **indefinida**.

```
> ./exemplo2.exe
```

```
35
```

```
0
```

```
Ctrl + c mata o processo, interrompendo o laço infinito
```

Para executar este arquivo com $a = 15, b = 15$ e saída $c = 1, a = 0$.

```
> ./exemplo2.exe
```

```
15
```

```
15
```

```
1
```

```
0
```

Exemplo 3 – Soma de inteiros

	15	#include <stdio.h>
	16	
	17	int main(void)
	18	{
	19	// Declarações locais
	20	int num = 0;
	21	int soma = 0;
	22	
1	soma \leftarrow 0	23 soma = 0;
2	leia o inteiro <i>número</i>	24 scanf("%d", &num);
3		25
4	enquanto <i>número</i> \neq 0	26 while (num != 0) {
	faça	27 soma = soma + num;
5	soma \leftarrow soma+número	28 scanf("%d", &num);
6	leia o inteiro <i>número</i>	29 } // while
7	fim enqto	30
8	escreva	31 printf("A soma dos números é %d\n",soma)
	"A soma dos números é",	;
	soma	32 return 0;
		33 } // main

Exemplo 3 – compilando e executando

Para compilar e gera o executável:

```
> gcc -std=c11 -c exemplo3.c  
> gcc exemplo3.o -o exemplo3.exe
```

Para executar este arquivo com a sequência: 12 23 34 0, faça

```
> ./exemplo3.exe  
12  
23  
34  
0  
A soma dos números é 69
```

Exemplo 4 – paridade

```
1 leia o número natural  $n$ 
2  $r \leftarrow \text{RESTO}(n \div 2)$ 
3
4 se  $r = 0$  então
5   | escreva "número é par"
6 senão
7   | escreva "número é ímpar"
8 fim se
```

```
9 #include <stdio.h>
10
11 int main(void)
12 {
13     // Declarações locais
14     unsigned int n = 0;
15     unsigned int r = 0;
16
17     scanf("%u", &n);
18     r = n % 2;
19
20     if (r == 0) {
21         printf("número é par\n");
22     } else {
23         printf("número é ímpar\n");
24     } // else
25
26     return 0;
27 } // main
```

Exemplo 4 – compilando e executando

Para compilar e gerar o executável:

```
> gcc -std=c11 -c exemplo4.c  
> gcc exemplo4.o -o exemplo4.exe
```

Para executar este arquivo com os números 2 5 0, digite:

```
> ./exemplo4.exe  
2  
número é par  
  
> ./exemplo4.exe  
5  
número é ímpar  
  
> ./exemplo4.exe  
0  
número é par
```


Exemplo 5 – equações do segundo grau

```
1  leia o número real a
2  se a = 0 então
3      escreva "a deve ser um
        número real diferente de
        zero"
4  senão
5      leia o número real b
6      leia o número real c
7       $\Delta \leftarrow b^2 - 4ac$ 
8      se  $\Delta \geq 0$  então // As
        raízes são reais
9           $x_1 \leftarrow \frac{-b + \sqrt{\Delta}}{2a}$ 
10          $x_2 \leftarrow \frac{-b - \sqrt{\Delta}}{2a}$ 
11         escreva "x1 = ", x1,
            "e x2 = ", x2
12     senão // As raízes são
        complexas
13         real  $\leftarrow \frac{-b}{2a}$ 
14         imag  $\leftarrow \frac{\sqrt{-\Delta}}{2a}$ 
15         escreva "x1 = ", real,
            "+ i(", imag, ")"
16         escreva "x2 = ", real,
            "- i(", imag, ")"
17     fim se
18 fim se
```

```
1
2 /**
3  * @file exemplo5.c
4  * @brief Encontra as raízes de uma equação de segundo grau.
5  *
6  * 0 programa lê três números em ponto flutuante a, b e c.
7  * 0 coeficiente a deve ser diferente de zero e, nesse caso,
8  * sempre haverá duas raízes reais ou complexas.
9  * Se a for zero, o programa imprime
10 * "a deve ser um número real diferente de zero" e termina.
11 * As raízes complexas, se existirem, são impressas no
12 * formato x + iy.
13 *
14 * @author Alexsandro Santos Soares
15 * @date 22/04/2018
16 * @bugs Nenhum conhecido.
17 */
18 #include <stdio.h>
19 #include <math.h>
20
21 int main(void)
22 {
23     // Declarações locais
24     float a = 0.0;
25     float b = 0.0;
26     float c = 0.0;
27     float delta = 0.0;
28     float x1 = 0.0;
29     float x2 = 0.0;
30     float real = 0.0;
31     float imag = 0.0;
```

Exemplo 5 – continuação

```
32
1  leia o número real a          33
2  se a = 0 então                34
3      escreva "a deve ser um número real diferente de zero" 35
4  senão
5      leia o número real b      36
6      leia o número real c      37
7       $\Delta \leftarrow b^2 - 4ac$     38
8      se  $\Delta \geq 0$  então // As 39
          raízes são reais      40
9           $x_1 \leftarrow \frac{-b + \sqrt{\Delta}}{2a}$  41
10          $x_2 \leftarrow \frac{-b - \sqrt{\Delta}}{2a}$  42
11         escreva "x1 = ", x1, "e 43
            x2 = ", x2          44
12     senão // As raízes são 45
        complexas              46
13         real  $\leftarrow \frac{-b}{2a}$  47
14         imag  $\leftarrow \frac{\sqrt{-\Delta}}{2a}$  48
15         escreva "x1 = ", real, "+ 49
            i(", imag, ")"      50
16         escreva "x2 = ", real, 51
            - i(", imag, ")"    52
17     fim se                    53
18 fim se                        54
                                55
```

```
scanf("%f", &a);
if (a == 0) {
    printf("a deve ser um número real
    diferente de zero\n");
} else {
    scanf("%f", &b);
    scanf("%f", &c);
    delta = b*b - 4*a*c;
    if (delta >= 0) {
        // As raízes são reais
        x1 = (-b + sqrt(delta)) / (2*a);
        x2 = (-b - sqrt(delta)) / (2*a);
        printf("x1 = %f e x2 = %f\n", x1, x2);
    } else {
        // As raízes são complexas
        real = -b / (2*a);
        imag = sqrt(-delta) / (2*a);
        printf("x1 = %f + i(%f)\n", real, imag);
        printf("x2 = %f - i(%f)\n", real, imag);
    } // else
} // else

return 0;
} // main
```

Exemplo 5 – compilando e executando

Para compilar e gerar o executável:

```
> gcc -std=c11 -c exemplo5.c  
> gcc exemplo5.o -o exemplo5.exe -lm
```

Para executar este arquivo com as entradas $a = 0$, $b = 1$, $c = 1$:

```
> ./exemplo5.exe  
0  
a deve ser um número real diferente de zero
```

Para executar este arquivo com as entradas $a = 1$, $b = 1$, $c = 1$:

```
> ./exemplo5.exe  
1  
1  
1  
x1 = -0.500000 + i(0.866025)  
x2 = -0.500000 - i(0.866025)
```

Exemplo 5 – compilando e executando

Para executar este arquivo com as entradas $a = 1$, $b = 2$, $c = 1$:

```
> ./exemplo5.exe  
1  
2  
1  
x1 = -1.000000 e x2 = -1.000000
```

Números complexos

- Desde 1999, a linguagem C possui suporte para números complexos.
- Vamos reescrever o programa anterior para usar os novos recursos presentes em C.
- Note que no código a seguir precisamos incluir a diretiva `#include <complex.h>` para usar as operações definidas sobre números complexos. Por exemplo:
 - `csqrt` retorna a raiz quadrada de números reais, incluindo os negativos, na forma de um número complexo.
 - `creal` retorna a parte real de um número complexo.
 - `cimag` retorna a parte imaginária de um número complexo.

Exemplo 6 – resolução de equação de segundo grau com números complexos

```
18 #include <stdio.h>
19 #include <complex.h>
20
21 int main(void)
22 {
23     // Declarações locais
24     float a = 0.0;
25     float b = 0.0;
26     float c = 0.0;
27     float delta = 0.0;
28     float complex x1 = 0.0;
29     float complex x2 = 0.0;
30
31     scanf("%f", &a);
32     if (a == 0) {
33         printf("a deve ser um número real diferente de zero\n");
34     } else {
35         scanf("%f", &b);
36         scanf("%f", &c);
37         delta = b*b - 4*a*c;
38
39         x1 = (-b + csqrt(delta)) / (2*a);
40         x2 = (-b - csqrt(delta)) / (2*a);
41         printf("x1 = %.3f + i(%.3f)\n", creal(x1), cimag(x1));
42         printf("x2 = %.3f + i(%.3f)\n", creal(x2), cimag(x2));
43     } // else
44
45     return 0;
46 } // main
```

Exemplo 6 – compilando e executando

Para compilar e gera o executável:

```
> gcc -std=c11 -c exemplo6.c  
> gcc exemplo6.o -o exemplo6.exe -lm
```

Para executar este arquivo com as entradas $a = 1$, $b = 1$, $c = 1$:

```
> ./exemplo6.exe  
1  
1  
1  
x1 = -0.500 + i(0.866)  
x2 = -0.500 + i(-0.866)
```

Exemplo 7 – classificação de triângulos

```
1 leia o número real x
2 leia o número real y
3 leia o número real z
4
5 se x < y + z E y < x + z E z < x + y então
6     se x = y E y = z então
7         escreva "Triângulo equilátero"
8     senão
9         se x = y Ou x = z Ou y = z
10            então
11                escreva "Triângulo isósceles"
12            senão
13                se x ≠ y Ou x ≠ z Ou y ≠ z
14                    então
15                        escreva "Triângulo escaleno"
16                    fim se
17                fim se
18            fim se
19 senão
20     escreva "Essas medidas não formam um triângulo"
21 fim se
```

```
18 #include <stdio.h>
19
20 int main(void)
21 {
22     float x = 0.0; float y = 0.0; float z = 0.0;
23
24     scanf("%f", &x);
25     scanf("%f", &y);
26     scanf("%f", &z);
27
28     if (x < y + z && y < x + z && z < x + y) {
29         if (x == y && y == z) {
30             printf("Triângulo equilátero\n");
31         } else if (x == y || x == z || y == z) {
32             printf("Triângulo isósceles\n");
33         } else if (x != y && x != z && y != z) {
34             printf("Triângulo escaleno\n");
35         } // if
36     } else {
37         printf("Essas medidas não formam um triângulo\n");
38     } // else
39
40     return 0;
41 } // main
```


Exemplo 7– compilando e executando

Para compilar e gerar o executável:

```
> gcc -std=c11 -c exemplo7.c  
> gcc exemplo7.o -o exemplo7.exe
```

Para executar este arquivo com $x = 1, y = 3, z = 1$:

```
> ./exemplo7.exe  
1  
3  
1  
Essas medidas não formam um triângulo
```

Para executar este arquivo com $x = 3, y = 3, z = 3$:

```
> ./exemplo7.exe  
3  
3  
3  
Triângulo equilátero
```

Exemplo 7– compilando e executando

Para executar este arquivo com $x = 3, y = 4, z = 3$:


```
> ./exemplo7.exe  
3  
4  
3  
Triângulo isósceles
```

Para executar este arquivo com $x = 3, y = 4, z = 5$:

```
> ./exemplo7.exe  
3  
4  
5  
Triângulo escaleno
```

Exemplo 8 – cálculo do peso ideal

	13	#include <stdio.h>
	14	
	15	int main(void)
	16	{
1	leia o número inteiro <i>a</i>	17 int a = 0.0;
2	leia o caracter <i>s</i>	18 char s = ' ';
		19 float peso = 0.0;
3		20
4	se <i>s</i> = 'm' então // Pessoa do	21 scanf("%d", &a);
	sexo masculino	22 scanf(" %c", &s); // descarta todos os brancos após %d,
5	peso ← $48 + 1.1 \times (a - 150)$	depois lê um char
6	escreva "Peso ideal =",	23
	peso	24 if (s == 'm') {
7	senão // Pessoa do sexo	25 // Pessoa do sexo masculino
	feminino	26 peso = $48 + 1.1 \times (a - 150)$;
8	se <i>s</i> = 'f' então	27 printf("Peso ideal = %.1f\n", peso);
9	peso ←	28 } else if (s == 'f') {
	$45 + 0.9 \times (a - 150)$	29 // Pessoa do sexo feminino
10	escreva "Peso ideal	30 peso = $45 + 0.9 \times (a - 150)$;
	=", peso	31 printf("Peso ideal = %.1f\n", peso);
11	senão	32 } else {
12	escreva "O sexo deve	33 printf("O sexo deve ser indicado por 'm' ou 'f'.\n");
	ser indicado por	34 } // else
	'm' ou 'f'."	35
13	fim se	36 return 0;
14	fim se	37 } // main

Obs: na linha 22, ao ler um caracter com `scanf` foi necessário usar " %c", com um espaço no início. A leitura de um inteiro na linha 21 deixa o caracter da tecla  no *buffer* do teclado.

Exemplo 8 – compilando e executando

Para compilar e gerar o executável:

```
> gcc -std=c11 -c exemplo8.c  
> gcc exemplo8.o -o exemplo8.exe
```

Para executar este arquivo com $a = 175$, $s = m$:

```
> ./exemplo8.exe  
175  
m  
Peso ideal = 75.5
```

Para executar este arquivo com $a = 154$, $s = f$:

```
> ./exemplo8.exe  
154  
f  
Peso ideal = 48.6
```

Para executar este arquivo com $a = 175$, $s = i$:

```
> ./exemplo8.exe  
175  
i
```

\square sexo deve ser indicado por 'm' ou 'f'

Exemplo 9 – cálculo do fatorial

```
12 #include <stdio.h>
13
14 int main(void)
15 {
16     unsigned int n = 0;
17     unsigned int i = 0;
18     unsigned int fat = 0;
19
20     scanf("%u", &n);
21
22     i = 0;    // contador de iterações
23     fat = 1; // acumula o valor do fatorial
24     while (i < n) {
25         i = i + 1;
26         fat = fat * i;
27     } // while
28     printf("Fatorial(%u) = %u\n", n, fat);
29
30     return 0;
31 } // main
```

```
1 leia o número natural  $n$ 
2  $i \leftarrow 0$  // contador de iterações
3  $\text{fat} \leftarrow 1$  // acumula o fatorial
4 enquanto  $i < n$  faça
5     |  $i \leftarrow i + 1$ 
6     |  $\text{fat} \leftarrow \text{fat} \times i$ 
7 fim enqto
8 escreva "Fatorial(",  $n$ , ")
   = ", fat
```

Exemplo 9 – compilando e executando

Para compilar e gerar o executável:

```
> gcc -std=c11 -c exemplo9.c  
> gcc exemplo9.o -o exemplo9.exe
```

Para executar este arquivo com $n = 0$:

```
> ./exemplo9.exe  
0  
Fatorial(0) = 1
```

Para executar este arquivo com $n = 1$:

```
> ./exemplo9.exe  
1  
Fatorial(1) = 1
```

Para executar este arquivo com $n = 5$:

```
> ./exemplo9.exe  
5  
Fatorial(5) = 120
```

Exemplo 10 – soma da progressão aritmética

```
1  leia o número real  $a_0$            // primeiro termo
2  leia o número inteiro  $n$          // número de termos
3  leia o número real  $r$              // razão
4   $i \leftarrow 0$                      // contador de iterações
5  soma  $\leftarrow 0$                   // acumula a soma
6   $a_i \leftarrow a_0$                 // o  $i$ -ésimo termo
7  enquanto  $i < n$  faça
8      soma  $\leftarrow$  soma +  $a_i$ 
9       $i \leftarrow i + 1$ 
10      $a_i \leftarrow a_i + r$ 
11 fim enqto
12 escreva "Soma da P.A =", soma
```

```
13 #include <stdio.h>
14
15 int main(void)
16 {
17     float a0 = 0.0;    // primeiro termo
18     int n = 0;         // número de termos
19     float r = 0.0;     // razão
20     int i = 0;         // contador de iterações
21     float soma = 0.0;  // acumula a soma
22     float ai = 0.0;    // o  $i$ -ésimo termo da PA
23
24     scanf("%f", &a0);
25     scanf("%d", &n);
26     scanf("%f", &r);
27
28     i = 0;
29     soma = 0.0;
30     ai = a0;
31     while (i < n) {
32         soma = soma + ai;
33         i = i + 1;
34         ai = ai + r;
35     } // while
36     printf("Soma da P.A. = %.3f\n", soma);
37
38     return 0;
39 } // main
```

Exemplo 10 – compilando e executando

Para compilar e gerar o executável:

```
> gcc -std=c11 -c exemplo10.c  
> gcc exemplo10.o -o exemplo10.exe
```

Para executar este arquivo com $a_0 = 2, n = 0, r = 2$:

```
> ./exemplo10.exe  
2  
0  
2  
Soma da P.A. = 0.000
```

Para executar este arquivo com $a_0 = 2, n = 1, r = 2$:

```
> ./exemplo10.exe  
2  
1  
2  
Soma da P.A. = 2.000
```


Exemplo 10 – executando

Para executar este arquivo com $a_0 = 2, n = 5, r = 2$:

```
> ./exemplo10.exe
```

```
2
```

```
5
```

```
2
```

```
Soma da P.A. = 30.000
```

Para saber mais

- Ascencio, A. F. G & Campos, E. A. V. *Fundamentos de programação de computadores: algoritmos, Pascal, C/C++ e Java*. 2. ed. São Paulo: Pearson Prentice Hall, 2007.
- *Doxygen: Main Page*. Disponível em <http://www.stack.nl/~dimitri/doxygen/>
- Marcelo Jo. *Documentação de código - Doxygen*. Disponível em <https://www.embarcados.com.br/documentacao-de-codigo-doxygen/>

Fontes

- Ascencio, A. F. G & Campos, E. A. V. *Fundamentos de programação de computadores: algoritmos, Pascal, C/C++ e Java*. 2. ed. São Paulo: Pearson Prentice Hall, 2007.