

Algoritmos e Programação de Computadores

Linguagem e Técnica de Programação - C

** Todos os direitos reservados nos termos da Lei Nº 9.610/98.*

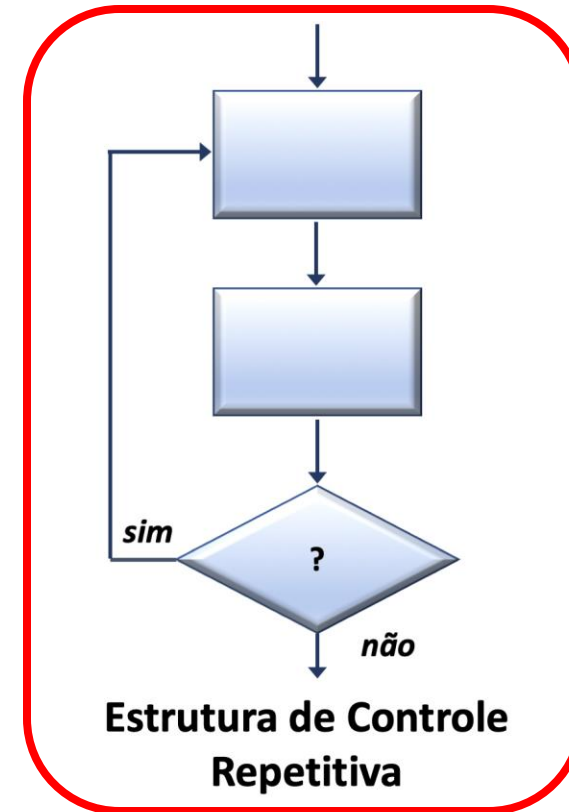
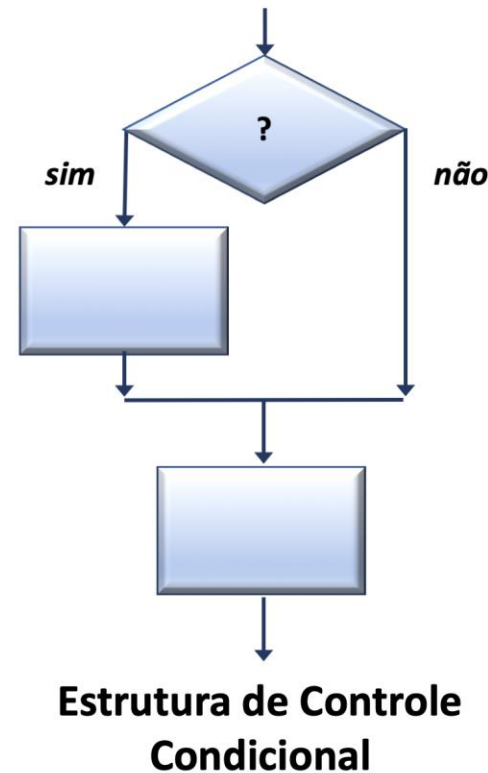
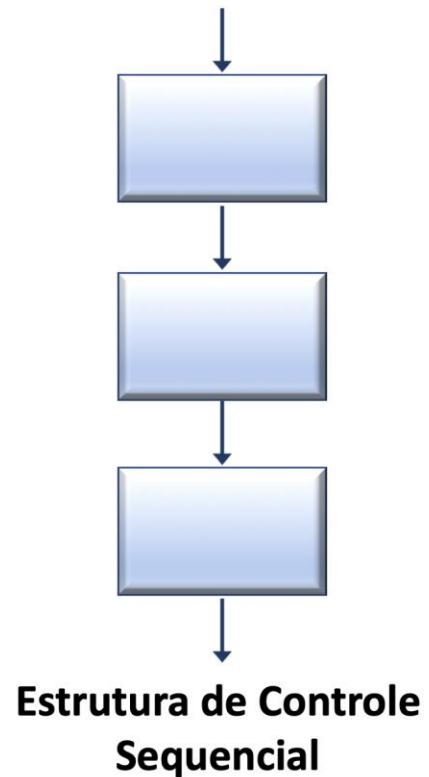
Programa da Disciplina

- **Organização básica de computadores**
- **Algoritmos**
- **Introdução a linguagem de programação estruturada**
- **Tipos de dados, identificadores, constantes e variáveis**
- **Comandos de entrada/saída e operadores**
- **Estruturas de decisão (*if, switch*)**
- **Estruturas de repetição (*for, while, do-while*)**
- **Avaliação P1**
- **Vetores e matrizes**
- ***Strings* e estruturas**
- **Modularização de programas**
- **Ponteiros e alocação dinâmica de memória**
- **Avaliação P2**

Sumário (Estruturas de Repetição)

- Laço for
(incremento / decremento, múltiplas expressões, com funções, omitindo o corpo do laço, omitindo expressões, encadeado)
- Laço while
(encadeado, término numérico, término caractere)
- Laço do-while
(exemplo)
- Os comandos *break* e *continue*

Programação Estruturada

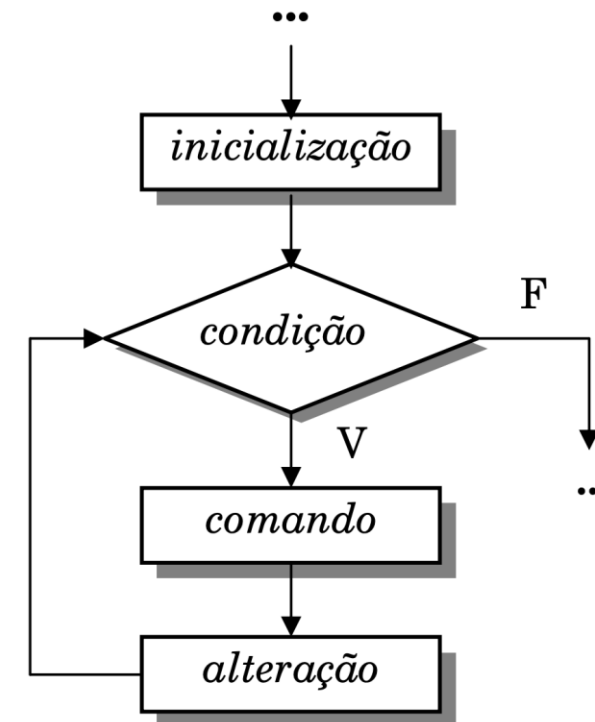


A estrutura de repetição permite que uma sequência de comandos seja executada repetidamente até que uma determinada condição de interrupção seja satisfeita.

O laço (*loop*) *for*

Sintaxe:

```
for (inicialização; condição; alteração)  
{  
    instruções;  
}
```



Usado principalmente quando é necessário repetir algo um número fixo de vezes.

O laço (*loop*) *for*

Sintaxe:

```
for (inicialização; condição; alteração)
{
    instruções;
}
```

As três expressões podem ser compostas por quaisquer instruções válidas em C e devem ser separadas por “;”.

- **inicialização:** executada uma única vez antes do início do laço.
 - Em geral é uma atribuição a uma variável de controle.
- **condição:** avaliada toda vez que o laço *for* iniciado ou reiniciado. Se verdadeira, o corpo do laço é executado. Se falsa, o laço é encerrado e o controle passa para a instrução seguinte.
 - Expressão lógica que controla o laço.
- **alteração:** executada sempre ao final de cada repetição do laço.
 - Define a forma com que a variável de controle será alterada a cada repetição do laço.

O laço (*loop*) *for* – incremento / decremento

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     for (num = 0; num <= 5; num++)
6.         printf (" %d ", num);
7.     return 0;
8. }
```

0 1 2 3 4 5

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     for (num = 5; num >= 0; num--)
6.         printf (" %d ", num);
7.     return 0;
8. }
```

5 4 3 2 1 0

O laço *(loop)* for – múltiplas expressões

```
1. #include <stdio.h>
2. int main()
3. {
4.     int x, y;
5.     for (x = 0, y = 0; x + y < 100; x += 1, y += 1)
6.         printf ("%d ", x + y);
7.     return 0;
8. }
```

Cada cláusula pode conter várias expressões!

- As múltiplas expressões são separadas por “,”.
- Elas são lidas da esquerda para a direita.

```
0  2  4  6  8  10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54
56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98
```


O laço (*loop*) *for* – associado a outros comandos

```
1. #include <stdio.h>
2. int main()
3. {
4.     int x, y;
5.     for (x = 0, y = 0; x + y < 100; x += 1, y += 1)
6.     {
7.         printf (" %2d ", x + y);
8.         if (x % 10 == 0)
9.             printf ("\n");
10.    }
11.    return 0;
12. }
```

Podemos usar qualquer comando associado a um *for*!

0									
2	4	6	8	10	12	14	16	18	20
22	24	26	28	30	32	34	36	38	40
42	44	46	48	50	52	54	56	58	60
62	64	66	68	70	72	74	76	78	80
82	84	86	88	90	92	94	96	98	

Linhas 6 a 10 podem ser substituídas por:
`printf ("%2d %s", x+y, (x+y)%20 == 0?"\n":"");`

O laço *(loop)* for – com caracteres

```
1. #include <stdio.h>
2. int main()
3. {
4.     char ch;
5.     for (ch = 'a'; ch < 'e'; ch++)
6.         printf (" O valor ASCII de %c é %d.\n", ch, ch);
7.     return 0;
8. }
```

Podemos usar caracteres ao invés de valores numéricos!

```
O valor ASCII de a é 97.
O valor ASCII de b é 98.
O valor ASCII de c é 99.
O valor ASCII de d é 100.
```

O laço (*loop*) *for* – com funções


```
1. #include <stdio.h>
2. int main()
3. {
4.     char ch;
5.     printf ("A cada letra lida será exibida a sua sucessora. \n");
6.     printf ("Pressione 'x' para sair. \n");
7.     for (ch = getchar(); ch != 'x'; ch = getchar())
8.         printf ("%c", ch+1);
9.     return 0;
10. }
```

Podemos chamar funções em qualquer expressão do laço!

```
A cada letra lida será exibida a sua sucessora.
Pressione 'x' para sair.
a
b
x
```

O laço (*loop*) *for* – omitindo o corpo do laço

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.      char ch;
5.      printf ("A cada letra lida será exibida a sua sucessora. \n");
6.      printf ("Pressione 'x' para sair. \n");
7.      for ( ; (ch = getchar()) != 'x'; printf ("%c", ch+1))
8.          ;
9.      return 0;
10. }
```



Necessário para indicar o corpo vazio.

```
A cada letra lida será exibida a sua sucessora.
Pressione 'x' para sair.
a
b
x
```

Podemos omitir o corpo do laço!

Testar ch++ e ++ch

O laço *(loop)* *for* – omitindo expressões

```
1. #include <stdio.h>
2. int main()
3. {
4.     for ( ; ; )
5.         printf ("Loop infinito\n");
6.     return 0;
7. }
```

Podemos omitir qualquer uma das três expressões!

[illegible]

- Os separadores (“;”) devem ser mantidos.
- Se a expressão de teste for omitida, ela é considerada permanentemente verdadeira.

O laço *(loop)* for – exemplo tabuada

```
1. #include <stdio.h>
2. int main()
3. {
4.     int n, c;
5.     printf ("\n Digite um número entre 1 e 10: ");
6.     scanf ("%d", &n);
7.     for (c = 1; c <= 10; c++)
8.         printf ("\n %d x %2d = %2d", n, c, n * c);
9.     return 0;
10. }
```

```
Digite um número entre 1 e 10: 2

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

O laço (*loop*) *for* – encadeado

```
1. #include <stdio.h>
2. int main()
3. {
4.     int n, c;
5.     for (n = 2; n <= 9; n++)
6.     {
7.         printf ("\n");
8.         for (c = 1; c <= 10; c++)
9.             printf ("\n %d x %2d = %2d", n, c, n * c);
10.    }
11.    return 0;
12. }
```

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

4 x 1 = 4
4 x 2 = 8
```

Laços *for* podem estar encadeados!

O laço (*loop*) *for* – encadeado

```
1. #include <stdio.h>
2. int main()
3. {
4.     int c, n;
5.     for(n=2; n<=9; n++)
6.         for(printf("\n\n Tabuada do %d", n), c=1; c<=10; c++)
7.             printf("\n %d x %2d = %2d", n, c, n * c);
8.     return 0;
9. }
```

Laços *for* podem estar encadeados e ter múltiplas instruções!

Tabuada do 2

2	x	1	=	2
2	x	2	=	4
2	x	3	=	6
2	x	4	=	8
2	x	5	=	10
2	x	6	=	12
2	x	7	=	14
2	x	8	=	16
2	x	9	=	18
2	x	10	=	20

Tabuada do 3

3	x	1	=	3
3	x	2	=	6
3	x	3	=	9
3	x	4	=	12
3	x	5	=	15
3	x	6	=	18
3	x	7	=	21
3	x	8	=	24
3	x	9	=	27
3	x	10	=	30

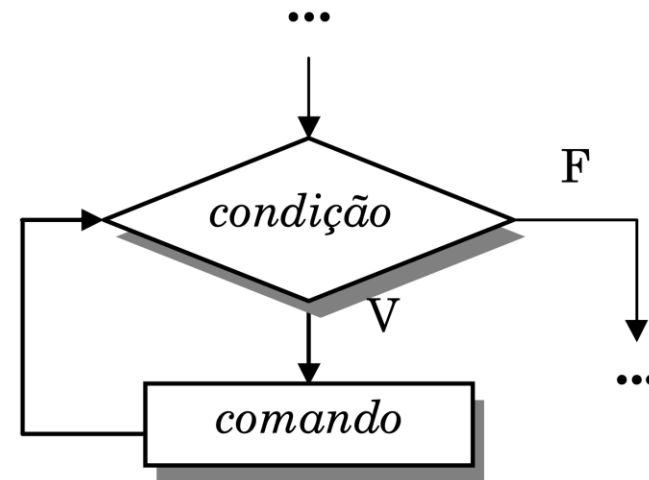
Tabuada do 4

4	x	1	=	4
4	x	2	=	8

O laço *while*

Sintaxe:

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



Usado para repetir algo com teste no início (pré-condição).

O laço *while*

Sintaxe:

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

Caso a condição nunca seja verdadeira
as instruções não serão executadas!

- **condição:** enquanto for verdadeira, o corpo do laço é executado. Se falsa, o laço é encerrado e o controle passa para a instrução seguinte.
 - Expressão lógica que controla o laço.
- **instruções:** pode ser uma única instrução ou um bloco de instruções.
 - Para um bloco de instruções, deve-se utilizar “{}”.

O laço *while* – exemplo

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     while (num <= 5)
6.     {
7.         printf (" %d ", num);
8.         num++;
9.     }
10.    return 0;
11. }
```

0 1 2 3 4 5

Usando o laço for:

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     for (num = 0; num <= 5; num++)
6.         printf (" %d ", num);
7.     return 0;
8. }
```

0 1 2 3 4 5

A estrutura do laço *while* possui os mesmos elementos da estrutura *for*, distribuídos de forma diferente.

O laço *while* – encadeado

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     long int resp;
6.     while (1)
7.     {
8.         printf (" Digite o número: ");
9.         scanf ("%d", &num);
10.        resp = 1;
11.        while (num > 1)
12.            resp *= num--;
13.        printf (" O fatorial é: %ld \n\n", resp);
14.    }
15.    return 0;
16. }
```

Condição sempre verdadeira
(*loop* infinito)!

```
Digite o número: 3
O fatorial é: 6

Digite o número: 4
O fatorial é: 24

Digite o número: █
```

O laço *while* – término numérico

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     printf ("Digite um número < 0 finaliza >\n");
6.     scanf ("%d", &num);
7.     while (num != 0)
8.     {
9.         printf ("Quadrado de %d = %d\n", num, num * num);
10.        scanf ("%d", &num);
11.    }
12.    return 0;
13. }
```

O número "0"
finaliza o loop!

```
Digite um número < 0 finaliza >
3
Quadrado de 3 = 9
6
Quadrado de 6 = 36
0
```

O laço *while* – término caractere

```
1. #include <stdio.h>
2. int main()
3. {
4.     char c;
5.     printf ("Digite uma letra < 'F' finaliza >\n");
6.     while (c != 'F')
7.         c = getchar();
8.     return 0;
9. }
```

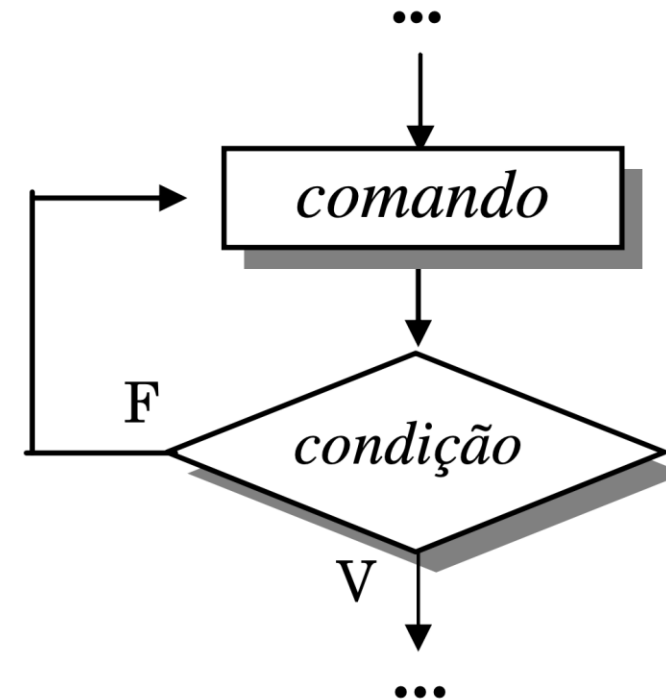
```
Digite uma letra < 'F' finaliza >
a
b
f
F
```

A letra "F"
finaliza o loop!

O laço *do-while*

Sintaxe:

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



Usado para repetir algo com teste no fim (pós-condição).

O laço *do-while*

Sintaxe:

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

As instruções serão executadas ao menos uma vez, mesmo que a condição nunca seja verdadeira!

- **condição:** enquanto for verdadeira, o corpo do laço é executado. Se falsa, o laço é encerrado e o controle passa para a instrução seguinte.
 - Expressão lógica que controla o laço.

- **instruções:** pode ser uma única instrução ou um bloco de instruções.
 - Para um bloco de instruções, deve-se utilizar “{}”.

OBS: tem um uso bastante efetivo na validação de dados de entrada, pelo teclado

O laço *do-while* (exemplo)

Laço *do-while*

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     do
6.     {
7.         printf (" %d ",num);
8.         num++;
9.     }
10.    while (num <= 5);
11.    return 0;
12. }
```

0 1 2 3 4 5

Laço *while*

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     while (num <= 5)
6.     {
7.         printf (" %d ", num);
8.         num++;
9.     }
10.    return 0;
11. }
```

0 1 2 3 4 5

Laço *for*

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     for (num = 0; num <= 5; num++)
6.         printf (" %d ", num);
7.     return 0;
8. }
```

0 1 2 3 4 5

O comando *break*

Às vezes é preciso **interromper um laço** antes que a sua condição torne-se falsa. Nessas ocasiões, podemos empregar o comando *break* que, quando não está associado ao *switch*, serve para parar uma repetição.

- Provoca a saída imediata do laço ou *switch*;
- Pode ser usado no corpo de qualquer estrutura de laço em C (*for*, *while* e *do-while*) ou *switch*;
- Se estiver em estrutura de laço encadeado, afetará apenas o laço que o contém e os laços internos a ele.

O comando *break*

Laço do-while

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     do
6.     {
7.         printf (" %d ",num);
8.         num++;
9.         if (num==3)
10.            break;
11.     }
12.     while (num <= 5);
13.     return 0;
14. }
```

0 1 2

Laço while

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     while (num <= 5)
6.     {
7.         printf (" %d ", num);
8.         num++;
9.         if (num==3)
10.            break;
11.     }
12.     return 0;
13. }
```

0 1 2

Laço for

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     for (num = 0; num <= 5; num++)
6.     {
7.         if (num==3)
8.            break;
9.         printf (" %d ", num);
10.    }
11.    return 0;
12. }
```

0 1 2

O comando *break* – números primos

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num, k;
5.     printf ("Digite um número natural: ");
6.     scanf ("%u", &num);
7.     for (k = 2; k <= num - 1; k++)
8.         if (num % k == 0)
9.             break;
10.    if (k == num)
11.        printf ("O número é primo");
12.    else
13.        printf ("O número não é primo");
14. }
```

```
Digite um número natural: 5
O número é primo
```

```
Digite um número natural: 6
O número não é primo
```

Um número é primo se for divisível apenas por um e por ele mesmo. Então, devemos testar se *num* é divisível por $k = 2, 3, 4, \dots, num-1$. Se *num* é divisível algum k , sendo $1 < k < num$, ele não é primo e o comando *break* pode ser acionado.

O comando *continue*

Às vezes é preciso **forçar o início da próxima iteração do laço**, descartando todas as instruções restantes do código atual. Nessas ocasiões, podemos empregar o comando *continue* que salta para a próxima iteração.

- Em laços *for* é feita a alteração da variável de controle e depois a verificação de condição;
- Em laços *while* e *do-while* a execução é desviada para a verificação de condição.

O comando *continue*

Laço do-while

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     do
6.     {
7.         if (num==3){
8.             num++;
9.             continue;}
10.        printf (" %d ",num);
11.        num++;
12.    }
13.    while (num <= 5);
14.    return 0;
15. }
```

Laço while

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num = 0;
5.     while (num <= 5)
6.     {
7.         if (num==3){
8.             num++;
9.             continue;}
10.        printf (" %d ", num);
11.        num++;
12.    }
13.    return 0;
14. }
```

Laço for

```
1. #include <stdio.h>
2. int main()
3. {
4.     int num;
5.     for (num = 0; num <= 5; num++)
6.     {
7.         if (num==3)
8.             continue;
9.         printf (" %d ", num);
10.    }
11.    return 0;
12. }
```

0 1 2 4 5



Diferença entre *for* x *while* x *do-while*

<i>for</i>	<i>while</i>	<i>do-while</i>
Estrutura compacta	Estrutura genérica	Estrutura genérica
Indicado quando se conhece o número de repetições.	Indicado quando o número de repetições é incerto.	Indicado quando o número de repetições é incerto.
As instruções podem não ser executadas.	As instruções podem não ser executadas.	As instruções são executadas pelo menos uma vez.

Atividades

Utilizando qualquer comando de [decisão](#) ou [repetição](#):

1) Escreva um programa que calcule e apresente na tela o quadrado de todos os números inteiros de **1** a **20**.

Resposta: 1, 4, 9, 16, 25, 36, ... 400

2) Escreva um programa que solicite ao usuário o primeiro elemento ***a*** de uma progressão aritmética (PA), a razão ***r*** desta PA, e o número ***n*** de elementos da PA. Apresente na tela todos os elementos da PA. Lembre-se que uma PA pode ser crescente ou decrescente.

Exemplos:

Para $a = 2$, $r = 3$ e $n = 6$ (2, 5, 8, 11, 14, 17)

Para $a = 2$, $r = -3$ e $n = 6$ (2, -1, -4, -7, -10, -13)

3) Escreva um programa que leia tantos números quanto o usuário desejar. Apresente na tela a quantidade de números lidos e a soma deles.

Exemplo:

Para os seguintes números informados pelo usuário: 2, 3, 5, 8 (número lidos = 4 e soma = 18)

4) Execute o teste de mesa para o slide 28. Apresente no pdf a evolução das variáveis na memória e saída do programa.

Referências

- **FORBELLONE, A. L. V., EBERSPACHER, H. F. Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados, 3ª Edição, São Paulo, Pearson Prentice Hall, 2005.**
- **SOUZA, M. A. F., GOMES, M. M., SOARES, M. V., CONCILIO, R. Algoritmos e Lógica de Programação, 3ª Edição, São Paulo, Cengage, 2019.**
- **PUGA, S., RISSETTI, G. Lógica de Programação e Estrutura de Dados, 3ª. Edição, Prentice Hall, 2016.**
- **DEITEL, H. M., DEITEL, P. J. C: Como Programar. LTC, 2011.**
- **ASCENCIO, A. F. G., CAMPOS, E. A. V. Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++. Pearson Prentice Hall, 2012.**
- **VAREJÃO, F. M. Introdução à Programação: Uma nova abordagem usando C. Campus, 2015.**
- **CELES, W., CERQUEIRA, R., RANGEL, J. L. Introdução a Estrutura de dados com Técnicas de Programação em C. Campus, 2016.**
- **MIZRAHI, V. V. Treinamento em Linguagem C – Curso Completo, 2ª Edição, Pearson Makron Books, 2008.**
- **SCHILDT, H., C Completo e Total, 3ª Edição, Makron Books, 1997.**