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

20 de novembro de 2022

Sumário

1 Processando a Informação: Cap. 4: Estruturas de Repetição (Laços)

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.

1.1 Sumário

- Revisão do capítulo anterios
- Quando usar repetições?
- Tipos de estruturas de repetição
- Laços aninhados
- Validação de dados com laços
- Interrupção da execução dos laços
- Recursão
- Revisão deste capítulo
- Exercícios

1.2 Revisão do capítulo anterior (Desvios Condicionais)

- No capítulo anterior foram apresentadas formas de construir código contendo estruturas condicionais simples ou compostas. Ou seja, desvios condicionais, da forma:
  - `se` algo for verdade, `então`
    - faça algo1,
  - `senão` # essa parte é opcional
    - faça algo2.
- Neste capítulo iremos abordar as **Estruturas de Repetição**, conhecidas também como **Laços** ou ***Loops***.

1.3 Quando usar repetições?

- As estruturas de repetição são recomendadas para quando um padrão de código é repetido várias vezes sequencialmente, apenas alterando-se o valor de uma ou mais variáveis entre os comandos repetidos.
- Veja no exemplo a seguir um pseudocodigo, para imprimir a tabuada de um número `t` entrado pelo usuário no formato:

Tabuada x (número de 1 a 10) = valor

```
Inteiro: t, n = 1;
t = leia("Qual a tabuada? ");
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
escreva(t + " x " + n + " = " + t * n);        n = n + 1;
```

- Embora o código não pareça extenso, é fácil imaginar uma situação onde tenhamos que repetir 100, 500, 1000 vezes um mesmo bloco de instruções.

1.4 Tipos de estruturas de repetição

1.4.1 Pseudocódigo

faça-enquanto

```
faça {
  comandos
} enquanto (condição);
```

enquanto-faça

```
enquanto (condição) faça {
  comandos
}
```

para

```
para variável = valor_inicial até valor_final, variável++, faça {
  comandos
}
```

para reverso

```
para variável = valor_final até valor_inicial, variável--, faça {
```

```
  comandos
}
```

- Note que no caso do `enquanto-faça` é necessário que a condição seja verdadeira para que os comandos presentes no bloco de execução sejam processados.
- Neste caso, se ao entrar no comando enquanto (*while*) a condição do teste for falsa, oposto ao `faça-enquanto`, o subprograma não será executado.
- Isto é, todo o código dentro do bloco do laço será pulado já na verificação inicial da condição no `enquanto-faça`, seguindo diretamente para a parte sequencial subsequente, similar ao que ocorre no se-então.
- A forma `faça-enquanto` é recomendada quando queremos que os comandos contidos no laço sejam executados ao menos uma vez, mesmo que a condição seja inicialmente falsa.
- O laço `para` é recomendado quando se sabe o número de iterações existes (quantas vezes o bloco dentro do laço será executado). Por exempo, no caso anterior do problema da Tabuada.

1.4.2 C/CPP/Java/JavaScript

do-while

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

while

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

for

```
for(v=0; v<10; v++) {
  comandos;
}
```

Em algumas linguagens de programação é possível omitir um ou todos os parâmetros, por exemplo: `for(;;) {...}`.

1.4.3 Pseudocódigo: Exemplo laço faça-enquanto.

```
Real: nota, média, acumulador=0, contador=0;
Caractere: resposta='lixo';

faça {
   nota = leia("Entre com uma nota: ");
   acumulador = acumulador + nota;
   contador = contador + 1;
```

```
   resposta = leia("Deseja continuar? (s/n): ");
} enquanto (resposta == 's');

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- Além do `contador`, o programa usa um `acumulador` (variável que acumula as notas digitadas).
- Repare que a condição `resposta == 's'` no `faça-enquanto` é falsa até que seja efetuada a leitura da variável resposta dentro do laço, em: `resposta = leia("Deseja continuar? (s/n): ");`
- Apenas do caso de o usuário entrar com o caractere `'s'`, o laço será repetido novamente.
- Isto quer dizer que o estado da condição é falso na primeira execução do código do laço.

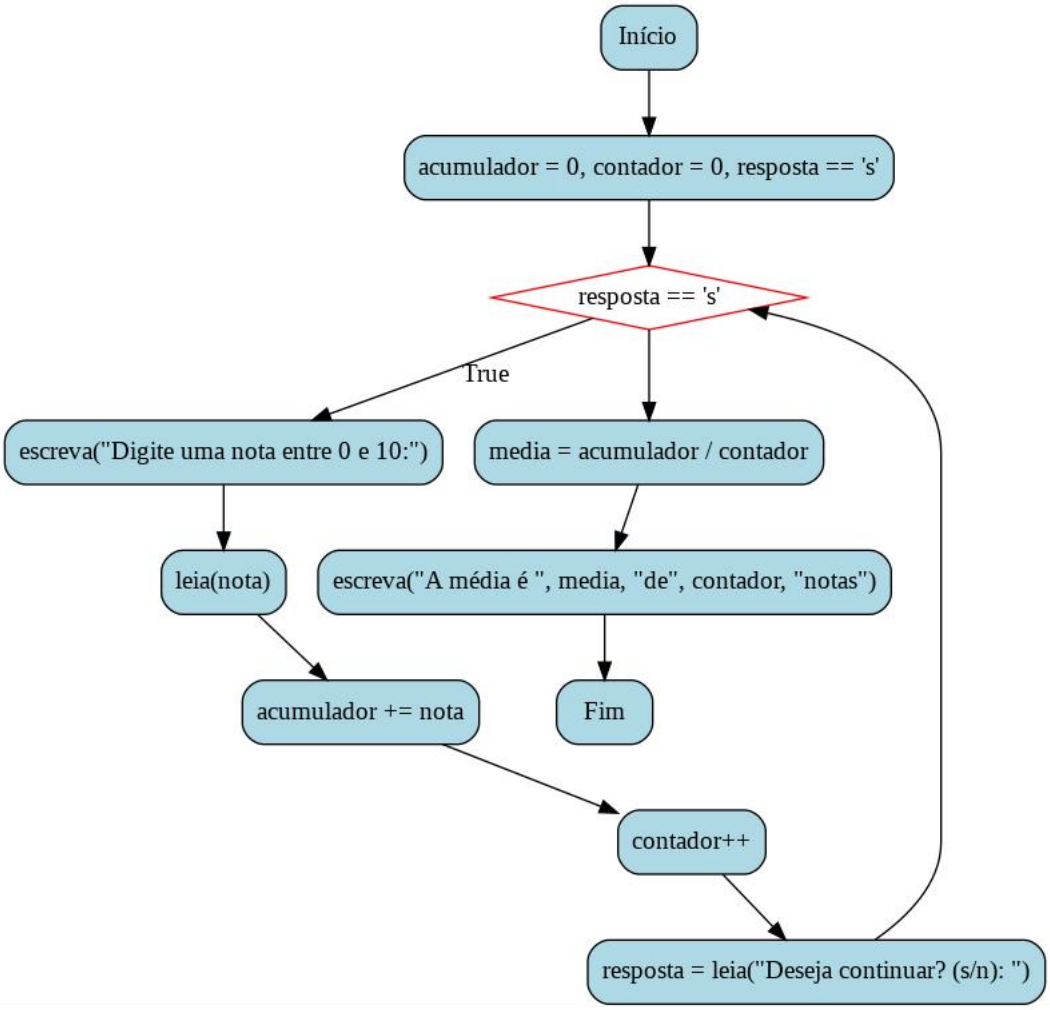
1.4.4 Pseudocódigo: Exemplo laço enquanto-faça.

```
Real: nota, média, acumulador=0, contador=0;
Caractere: resposta='s';

enquanto (resposta == 's') faça {
   nota = leia("Entre com uma nota: ");
   acumulador = acumulador + nota;
   contador = contador + 1;
   resposta = leia("Deseja continuar? (s/n): ");
}

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- Observe que no pseudocódigo anterior do `enquanto-faça` temos o mesmo resultado do `faça-enquanto`, pois a variável `resposta` é inicializada com `s`, satisfazendo a condição lógica e entrando no laço.
- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo):



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    acumulador = 0, contador = 0, resposta == 's';
    while (resposta == 's') {
      escreva("Digite uma nota entre 0 e 10:");
      leia(nota);
      acumulador += nota;
      contador++;
      resposta = leia("Deseja continuar? (s/n): ");
    }
    media = acumulador / contador;
    escreva("A média é ", media, "de", contador, "notas");
```

```
    Fim;
    '''
)
```

1.4.5 Pseudocódigo: Exemplo laço enquanto-faça INFINITO.

Considere uma alteração no código anterior para não fazer mais a pergunta `Deseja continuar? (s/n)`, mas entrando num laço `enquanto-faça` para ler e acumular 100 notas:

```
Real: nota, média, acumulador=0,contador=0;

enquanto (contador < 100) faça {
   nota = leia("Entre com uma nota: ");
   acumulador = acumulador + nota;
   # contador = contador + 1;
}

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- O que vai ocorrer ao escrever e rodar esse código em alguma linguagem de programação?
- Onde está o erro?
- **MUITO CUIDADO COM LAÇOS INFINITOS EM ATIVIDADES NO MOODLE+VPL!** Se a execução demorar mais que 1 minuto, provavelmente entrou em um laço infinito e terá que recarregar a página.
- Esta condição, onde a execução fica "presa" dentro do laço, é conhecida como ***DEADLOCK***.
- *Deadlocks* geram erro de finalização de programa, que executará eternamente, podendo travar o programa, o teclado e o mouse ou até mesmo o computador, neste caso, sendo necessário um ***RESET*** para sair do laço.

1.5 Validação de Dados

- Uma possível aplicação de laços é garantir que os dados entrados sejam válidos.
- **Validação de dados** é o nome dado à verificação dos valores de entrada, se os mesmos se encontram dentro dos limites previstos ou no formato adequado, notificando o usuário no caso de valores inválidos.
- O exemplo a seguir é o pseudocódigo apresentado anteriormente, incorporando a validação de dados de entrada (nota entre 0 e 10), indicando o erro e pedindo para o usuário entrar novamente o dado até que seja válido.

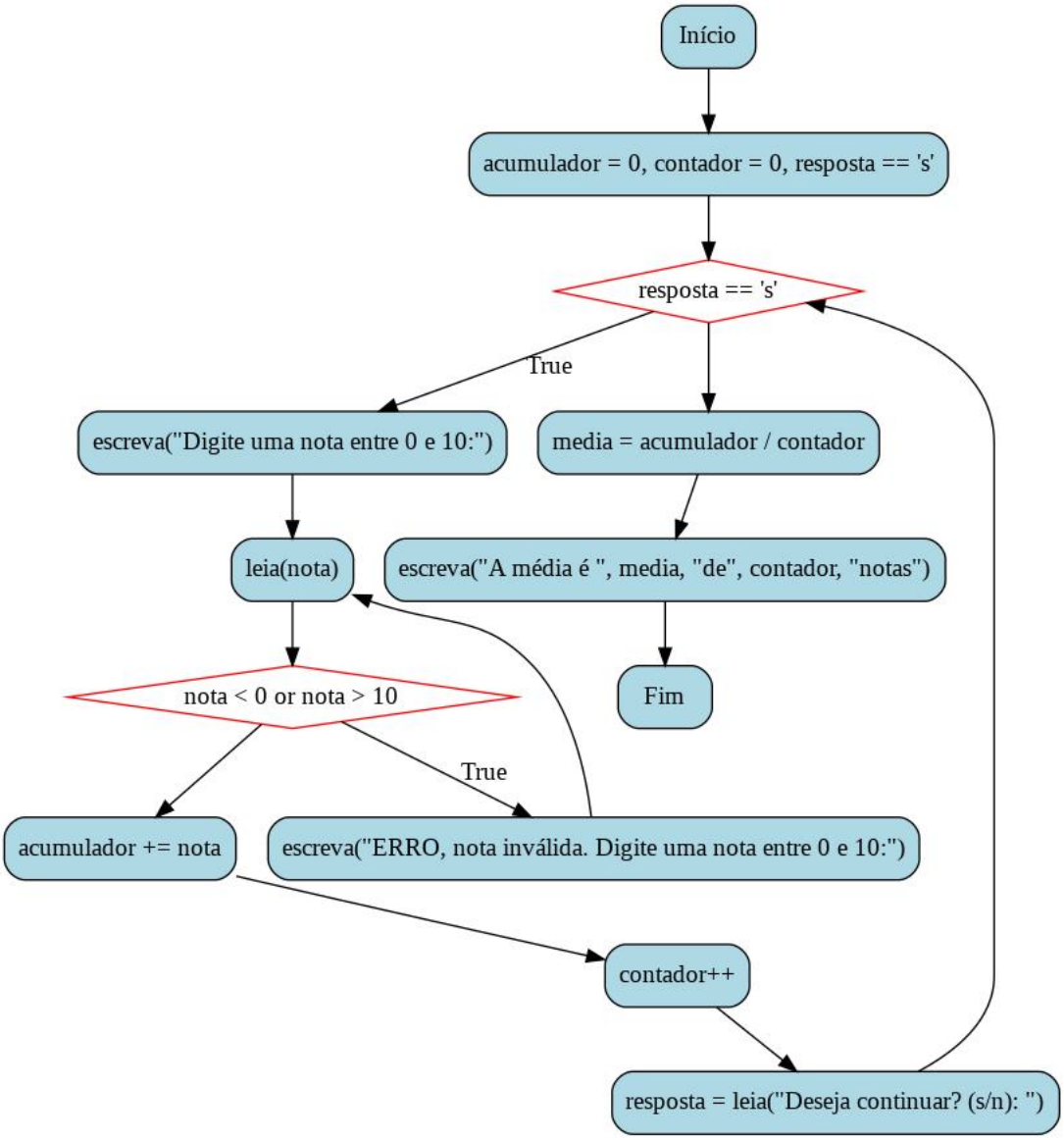
1.5.1 Pseudocódigo: Exemplo laço faça-enquanto, com validação

```
Real: nota, média, acumulador=0, contador=0;
Caractere: resposta='lixo';
```

```
faça {
  faça {
    nota = leia("Entre com uma nota entre 0 e 10: ");
    se (nota < 0 || nota > 10) então
      escreva("ERRO, nota inválida. Digital nota entre 0 e 10!")
  } enquanto (nota < 0 || nota > 10);
  acumulador = acumulador + nota;
  contador = contador + 1;
  resposta = leia("Deseja continuar? (s/n): ");
} enquanto (resposta == 's');

média = acumulador / contador;
imprima ("A média das " + contador + " notas é " + média);
```

- No pseudocódigo anterior, o segundo `faça-enquanto` aceita apenas notas entre 0 e 10. Caso contrário, escreve uma mensagem de erro e solicita nova nota.
- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo).
- Observar que essa biblioteca `txtoflow` não aceita `faça-enquanto`, assim como o Python.



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    acumulador = 0, contador = 0, resposta == 's';
    while (resposta == 's') {
      escreva("Digite uma nota entre 0 e 10:");
      leia(nota);
      while (nota < 0 or nota > 10) {
        escreva("ERRO, nota inválida. Digite uma nota entre 0 e 10:");
```

```
        leia(nota);
      }
      acumulador += nota;
      contador++;
      resposta = leia("Deseja continuar? (s/n): ");
    }
    media = acumulador / contador;
    escreva("A média é ", media, "de", contador, "notas");
    Fim;
    '''
)
```

1.6 Interrupção dos laços

- Algumas linguagens permitem interromper a execução do laço através do comando 'interromper' ou 'quebrar' (**break**).
- Isto pode ser útil caso se queira interromper o laço em algum evento específico.

1.7 Exemplo 01 - Ler 10 notas com validação

Considere um algoritmo para ler 10 notas válidas, entre 0 e 10, escrevendo no final a média nas notas válidas lidas.

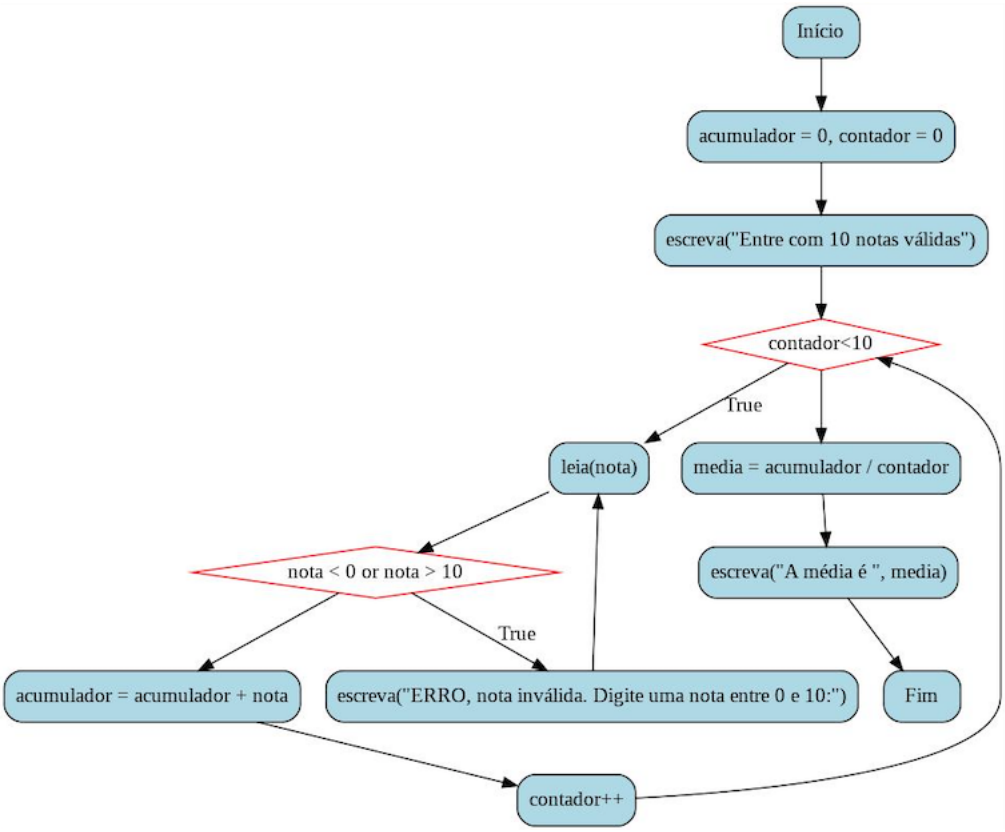
Pseudocódigo

```
Real: nota, média, acumulador=0,contador=0;

escreva("Entre com 10 notas válidas")
faça {
  faça {
    nota = leia();
  } enquanto (nota < 0 || nota > 10);
  acumulador = acumulador + nota;
  contador = contador + 1;
} enquanto (contador < 10);

média = acumulador / contador;
escreva("A média das " + contador + " notas é " + média);
```

- Ver Fluxograma abaixo (também experimente nessa ferramenta *online*: code2flow, copiando e colando o código em vermelho abaixo):



[ ]:
```
!apt-get install graphviz libgraphviz-dev pkg-config
!pip install txtoflow
```

[ ]:
```
from txtoflow import txtoflow

txtoflow.generate(
    '''
    Início;
    acumulador = 0, contador = 0;
    escreva("Entre com 10 notas válidas");
    while (contador<10) {
      leia(nota);
      while (nota < 0 or nota > 10) {
        escreva("ERRO, nota inválida. Digite uma nota entre 0 e 10:");
        leia(nota);
      }
      acumulador = acumulador + nota;
      contador++;
    }
    media = acumulador / contador;
    escreva("A média é ", media);
    Fim;
    '''
```

```
)
```

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 (pode incluir mais casos):

```
case=caso1
input=9.0
78.0
6.0
9
8
7
6
5
7
8
6
output=
A média das 10 notas é 7.1
case=caso2
input=9.0
78.0
-9.0
9876.0
9876.0
7.0
9
8
4
6
6
8
9
7
output=
A média das 10 notas é 7.3
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[ ]:
```
%%writefile cap4ex01.c
#include <stdio.h>

int main(void) {

  // ENTRADA DE DADOS e PROCESSAMENTO
  float acumulador = 0, nota, media;
```

```c
  int contador = 0;

  printf("Entre com 10 notas válidas\n");
  while (contador<10) {
    do {
        scanf("%f", &nota);
    } while (nota < 0.0 || nota > 10.0 );
    acumulador = acumulador + nota;
    contador++;
  }
  media = acumulador/contador;

  // SAÍDA
  printf("A média das %d notas é %.1f\n", contador, media);

  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap4ex01.c -o output
./output
```

1.8 Recursão

Este tópico de recursão (ou recursividade) é complementar ao livro, em sua primeira edição.

Como nos laços de repetição, a recursão tem como objetivo rodar trechos de códigos (agora encapsulados em métodos) várias vezes.

Além disso, análogo ao laço, **muito cuidado com o critério de parada, senão o código irá fazer infinitas chamadas recursivas, podendo travar o seu computador!**

Ou seja, o método recursivo teve ter pelo menos uma condicional e argumentos que variam nas chamadas recursivas que garantam a convergência (critério de parada).

As funções a seguir atendem a esses requisitos? Senão, como corrigir? O que elas fazem?

```c
int funcao_recursiva(int a) {
    if (a == 0) return a;
    return 1+funcao_recursiva(a-1);
}
int funcao_recursiva(int a) {
    if (a == 0) return a;
    return 1+funcao_recursiva(a-2);
}
```

Veja mais um exemplo a seguir.

1.9 Exemplo 02 - Fatorial

Considere um algoritmo para calcular o fatorial de n [ref].

Pseudocódigo

```
# MINHA FUNÇÃO RECURSIVA
função fatorial(recebe: inteiro n) retorna inteiro {
  se (n == 1) faça { # CRITÉRIO DE PARADA
    retorne 1
  }
  retorne n * fatorial(n-1);
}

Inteiro: n;
escreva("Entre com numero: " + n)
escreva("Fatorial de " + n + " é " + fatorial(n) );
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[16]:
```
%%writefile cap4ex02.c
#include <stdio.h>

int fatorial(int n) {
  long int f;
  printf("debug - antes  - n = %d\n",n);
  if (n == 1) f = 1; // CRITÉRIO DE PARADA!!!
  else // CHAMADA RECURSIVA, COM ALTERAÇÃO DO VALOR DO ARGUMENTO!!!
    f = n * fatorial(n-1);
  printf("debug - depois - n = %d; fatorial = %ld\n", n, f);
  return f;
}

int main(void) {
  // ENTRADA DE DADOS e PROCESSAMENTO
  int n;

  printf("Entre com n: ");
  scanf("%d", &n);

  // PROCESSAMENTO E SAÍDA
  printf("Fatorial de %d é %d\n", n, fatorial(n));

  return 0;
}
```

[17]:
```
%%shell
gcc -Wall -std=c99 cap4ex02.c -o output
./output
```

1.10 Exemplo 03 - Ler 10 notas, com recursão

Considere um algoritmo para ler 10 notas válidas utilizando recursão e calcular a média.

Pseudocódigo

```
# MINHA FUNÇÃO RECURSIVA
função lerNota(recebe: real acumulador, inteiro n) retorna real acumulador {
  Real: nota;
  se (n > 0) faça { # CRITÉRIO DE PARADA
    faça {
      nota = leia();
    } enquanto (nota < 0 || nota > 10);
    acumulador = lerNota(acumulador, n - 1); # CHAMADA RECURSIVA,
    # COM ALTERAÇÃO DO VALOR DO ARGUMENTO !!!
  }
  retorne acumulador + nota;
}

Real: média, acumulador=0, contador=10;

escreva("Entre com " + contador + " notas válidas")
acumulador = lerNota(acumulador, contador); # CHAMDADA RECURSIVA!!!

média = acumulador / contador;
escreva("A média das " + contador + " notas é " + média);
```

- Experimente essa ferramenta *online* para visualizar o fluxograma do código a seguir (copie o código e cole na ferramenta): code2flow.

[ ]:
```c
%%writefile cap4ex03.c
#include <stdio.h>

float lerNota(float acumulador, int n) {
  float nota;
  if (n > 0) { // CRITÉRIO DE PARADA!!!
    do {
      scanf("%f", &nota);
    } while (nota < 0.0 || nota > 10.0);
    // CHAMADA RECURSIVA, COM ALTERAÇÃO DO VALOR DO ARGUMENTO!!!
    acumulador = lerNota(acumulador, n - 1);
  }
  return acumulador + nota;
}

int main(void) {

  // ENTRADA DE DADOS e PROCESSAMENTO
  float acumulador = 0, media;
  int contador = 10;
```

```
  printf("Entre com %d notas válidas\n", contador);

  // CHAMADA RECURSIVA
  acumulador = lerNota(acumulador, contador);

  media = acumulador / contador; // MÉDIA

  // SAÍDA
  printf("A média das %d notas é %.1f\n", contador, media);

  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap4ex03.c -o output
./output
```

1.11 Exercícios

Ver notebook Colab no arquivo `cap4.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.

1.12 Revisão deste capítulo de Estruturas de Repetição (Laços)

- Quando usar repetições? > Quando existem instruções que se repentem.
- Tipos de estruturas de repetição:
  - Depende da linguagem. Algumas possibilidades:
    - `do-while` (não aceita em Python)
    - `while` (todas)
    - `for` (todas)
    - `repeat` (R)
  - Qual usar?
    - Depende da lógica ser implementada e da linguagem utilizada.
    - Se tiver um número fixo de iterações, geralmente se usa `for`.
- Laços aninhados
- Validação de dados com laços
  - Incluir um laço para verificar o valor lido.
- Interrupção da execução dos laços
  - Depende da linguagem, algumas possibilidades:
    - `break` - interrompe o laço
    - `continue` - não executa o final do laço
    - `exit` - aborta o laço e o programa!
- Outra forma de executar trechos de códigos várias vezes é encapsular em métodos recursivos.
  - Atenção com o critério de parada!
  - Atenção com os argumentos do método recursivo!

- Exercícios
- Revisão deste capítulo de Estruturas de Repetição (Laços)