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

8 de setembro de 2022

Sumário

1 Processando a Informação: Cap. 3: Desvios Condicionais

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

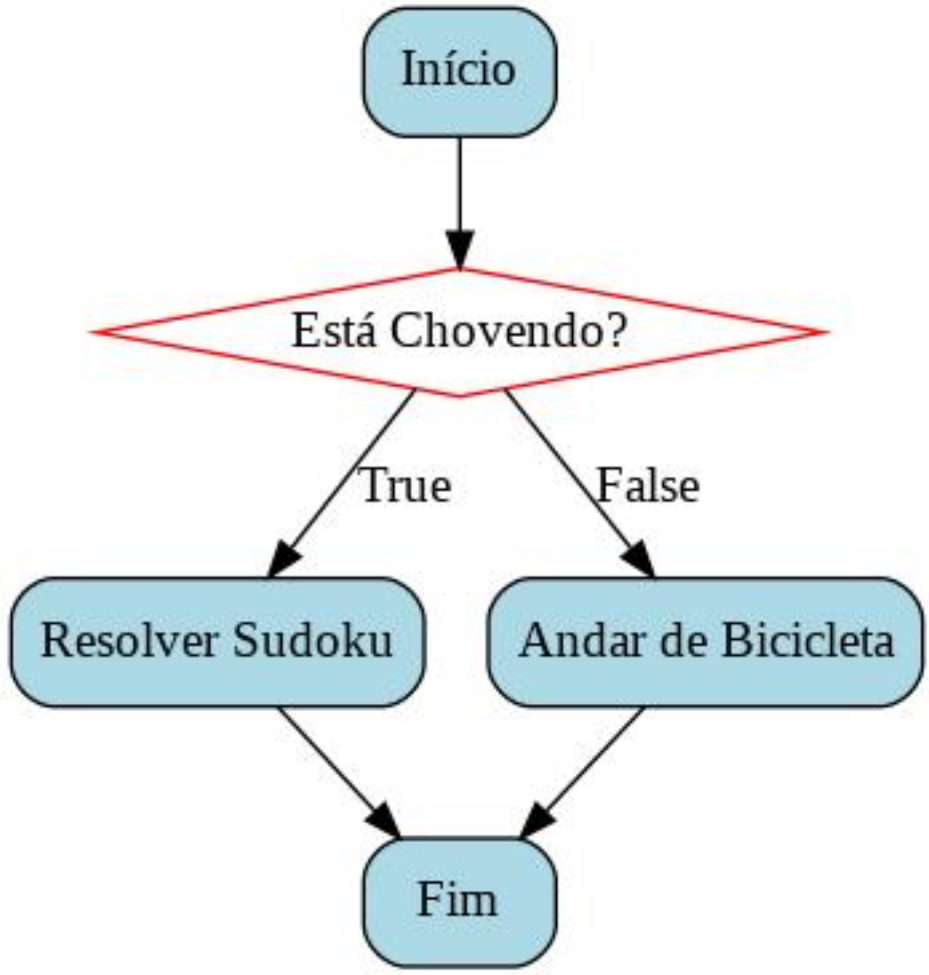
- Revisão do capítulo anterios
- O que é um desvio condicional?
- Condições com lógica booleana
- Desvios condicionais Simples e Compostos, com Encadeamentos
- Revisão deste capítulo
- Exercícios

1.2 Revisão do capítulo anterior (Organização de Código)

- No capítulo anterior foram apresentados formas de organização de códigos, utilizando comentários, tabulações, escopo de variáveis locais e globais, métodos (funções ou procedimentos) e o conceito de sistema de informação em partes:
  - **ENTRADA DE DADOS ⇒ PROCESSAMENTO DA INFORMAÇÃO ⇒ SAÍDA**
- Neste capítulo serão apresentados códigos com desvios condicionais, da forma:
  - `se` algo for verdade, `então`
    - faça algo1,
  - `senão` # essa parte é opcional
    - faça algo2.

1.3 O que é um Desvio Condional?

- O desvio condicional é a mais simples entre as estruturas lógicas não sequenciais em lógica de programação e fundamental para o entendimento de fluxo de código.
- A analogia básica com o processo de tomada de decisões ocorre quando imaginamos um cenário que proporciona duas possíveis alternativas de curso:
  - Se [condição] então faça [caminho caso verdadeiro] senão [caminho caso falso]
- Exemplos:
  - Se [está chovendo] então [resolver palavras cruzadas] senão [andar de bicicleta]
  - Se [é quarta] então [comer feijoada]
- 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;
    if (Está Chovendo?) {
        Resolver Sudoku;
    } else {
        Andar de Bicicleta;
    }
    Fim;
    '''
)
```

1.4 Condições com Lógica *Booleana*

- O resultado de um teste condicional sempre resultará em um valor *booleano*, isto é:
  - com dois resultados possíveis: **verdadeiro** ou **falso**
  - Por convensão: $True = 1$ e $False = 0$
- Portanto, para condições, sempre usaremos combinações de operadores lógicos e relacionais para verificar o estado das variáveis verificadas. O seguinte pseudocódigo exemplifica algumas condições:

```
se vai chover, então leve um guarda-chuva.

se é feriado, então fique em casa.

se estou atrasado e está chovendo, então chame um taxi.

se minha nota é menor que 5, então fiquei de recuperação.
```

- Note que para todas as condições acima, a resposta para a condição é sempre: verdadeiro ou falso.
- Caso a condição seja verdadeira, será executada a operação ou operações especificadas na sequência.
- Codificando-se, as condições tomam a forma `se (condição) { comandos }`, como exemplos:

```
se (vai_chover) { leve um guarda-chuva }

se (feriado) { fique em casa }

se (atrasado e chovendo) { chame um taxi }

se (nota<5) { escrever ‘‘ficou de Recuperação’’ }
```

Esse código anterior **fica melhor se usar a organização** apresentada no capítulo anterior:

```
se (vai_chover)
  leve um guarda-chuva

se (feriado)
  fique em casa

se (atrasado e chovendo)
  chame um taxi

se (nota<5)
  escrever ‘‘ficou de Recuperação’’
```

- Observe que foram substituídas as chaves `{` e `}`, que definem os **escopos** das condicionais, pela tabulação. Isso é possível quando o bloco tem apenas uma instrução.
- Relembrando o Cap. 1 - Fundamentos, podemos usar nas **condicionais**, variáveis *booleanas* e operadores:
  - **relacionais**: ∘ == (é igual a) ∘ != (é diferente de) ∘ > (é maior que) ∘ < (é menor que) ∘ >= (é maior ou igual a) ∘ <= (é menor ou igual a)
  - Além de **lógicos**: ∘ && (e), & (e bit-a-bit) ∘ || (ou), | (ou bit a bit) ∘ ! (não lógico ou complemento) ∘ ~ (complemento bit-a-bit) ∘ ^ (ou exclusivo ‘XOR’ bit-a-bit) ∘ « N (shift-left, adiciona N zeros à direita do número em binário) ∘ » N (shift-right, elimina N dígitos a direita do número em binário)
- Os operadores **lógicos**, como os **relacionais**, sempre resultarão **verdadeiro** (V) ou **falso** (F), porém os operandos também são *booleanos*.
- Para as condições contendo AND, OR e XOR (ou exclusivo), as **tabelas verdade** para os dois operandos à esquerda e à direita, com valores lógicos representados na primeira linha e primeira coluna, são:

&&	V	F
V	**V**	F
F	F	F

||	V	F
V	V	V
F	V	**F**

^	V	F
V	F	V
F	V	F

- Logo,
  - os operandos devem ser ambos verdadeiros para que a operação AND retorne verdadeiro,

  - ao menos um deles verdadeiro para que o OR retorne verdadeiro e
  - ambos diferentes para que o XOR (ou exclusivo ^) retorne verdadeiro.

- Estas expressões, quando combinadas, resultarão sempre em um valor *booleano* V ou F, que pode ser então introduzido em um desvio condicional visando à realização de um subprograma.
- Por exemplo, usando os valores X=1, Y=2 e Z=4, qual o resultado das expressões abaixo?

  1. (X>0 && Y<2)
  2. (Z>0 || Z<5 && Y==4)
  3. (X»1 ==0 || Y«2>100)
  4. (X!=0 && Y!=0 && Z<0)
  5. (X=1)

- Dada a precedência de operadores estudada anteriormente e a combinação apresentada acima, apenas as expressões 2 e 3 resultariam em verdadeiro, dado $Z > 0$, assim como $1 >> 1$ (*shift* à direita uma casa) em binário é 0; são ambas condições suficientes para que o resultado seja verdadeiro.

[ ]:
```
X, Y, Z =1, 2, 4
equacao1 = (X>0 and Y<2)
equacao2 = (Z>0 or Z<5 and Y==4)
equacao3 = (X>>1 ==0 or Y<<2>100)
equacao4 = (X!=0 and Y!=0 and Z<0)
#equacao5 = (X=1) # ERRO de sintaxe, deveria ser X==1
print(equacao1,equacao2,equacao3,equacao4)
```

[ ]:
```
# exemplo de uso de ">>"
# 6 = 1 1 0 em binário => 1*2^2 + 1*2^1 + 0*2^0
6>>1 # = 3 = 0 1 1 em binário
```

- É importante ressaltar a diferença entre os operadores de comparação `==`, com leitura `é igual à`, e de atribuição `=`, tendo a leitura `recebe o valor de`. Neste aspecto, a expressão 5 está incorreta, já que o operador de atribuição não faz sentido quando usado desta forma.

1.5 Desvios Condicionais Simples e Compostos

```
se (condição) então faça
    Comandos

Volta para a parte sequencial
```

```
se (condição) então faça
    Comandos
senão faça
    Comandos

Volta para a parte sequencial
```

```
if (condição) {
    Comandos
}
```

```
if (condição) {
    Comandos
} else {
    Comandos
}
```

Exemplo 01 - Uso de Condicionais Simples

- Digamos que, como exemplo, desejamos calcular as raízes da equação de segundo grau usando a função `delta()` introduzida no capítulo anterior.
- Sabemos que as raízes dependem do sinal do Δ.
- Logo, a solução de uma equação do segundo grau se dá resolvendo as seguintes condições:
  1. Se $\Delta < 0$, x não possui raízes reais;
  2. Se $\Delta = 0$, x possui duas raízes reais idênticas;
  3. Se $\Delta > 0$, x possui duas raízes reais e distintas;
  4. Calcule as raízes, se existirem, usando as equações:

$$x1 = -b + \frac{\sqrt{\Delta}}{2a}$$

$$x2 = -b - \frac{\sqrt{\Delta}}{2a}$$

- Veja uma solução em pseudocódigo:

```
# a função é definida a seguir
função delta(a, b, c )
  retorne b * b - 4 * a * c

# programa principal
# ENTRADA DE DADOS
escreva("Calcula as raízes de equação de 2º grau: ax2 + bx + c")
real a = leia("Entre com o primeiro termo 'a': ")
real b = leia("Entre com o segundo  termo 'b': ")
real c = leia("Entre com o terceiro termo 'c': ")

# PROCESSAMENTO E SAÍDA
real d = delta(5, -2, 4)
escreva(''O delta é '' + valor)
se (d < 0)
  escreva("A equação não possui raízes reais")
se (d == 0)
  escreva("A raíz é " + (-b + raíz(d)/2*a))
```

```
se (d > 0)
  escreva("As raízes são x1=" + (-b - raíz(d)/2*a)) + " e x2="  +
  (-b + raíz(d)/2*a))
```

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=3
1
4
output=
O delta é  -47.0
A equação não possui raízes reais.
case=caso2
input=4
6
2
output= O delta é 4.0
Raízes: -10.0 e -2.0.
```

[ ]:
```
%%writefile cap3ex01.c
#include <stdio.h>
#include <math.h>
float delta(float a, float b, float c) {
  float d = b*b-4*a*c;
  return d;
}

float leia() {
  float valor;
  printf("Entre com um valor: ");
  scanf("%f", &valor);
  return valor;
}

int main(void) {

  // ENTRADAS
  float a, b, c;
  a = leia();
  b = leia();
  c = leia();

  // PROCESSAMENTO e SAÍDA
  double d = (double) delta(a, b, c);
  printf("Delta = %.1f\n", d);
  if (d < 0) {
```

```
      printf("A equação não possui raízes reais");
  }
  if (d == 0) {
      printf("Raíz: %.1f",(-b + sqrt(d) / 2 * a));
  }
  if (d > 0) {
      printf("Raíz: %.1f e %.1f ",(-b - sqrt(d) / 2 * a), (-b + sqrt(d)␣
↪/ 2 * a));
  }
  return 0;
}
```

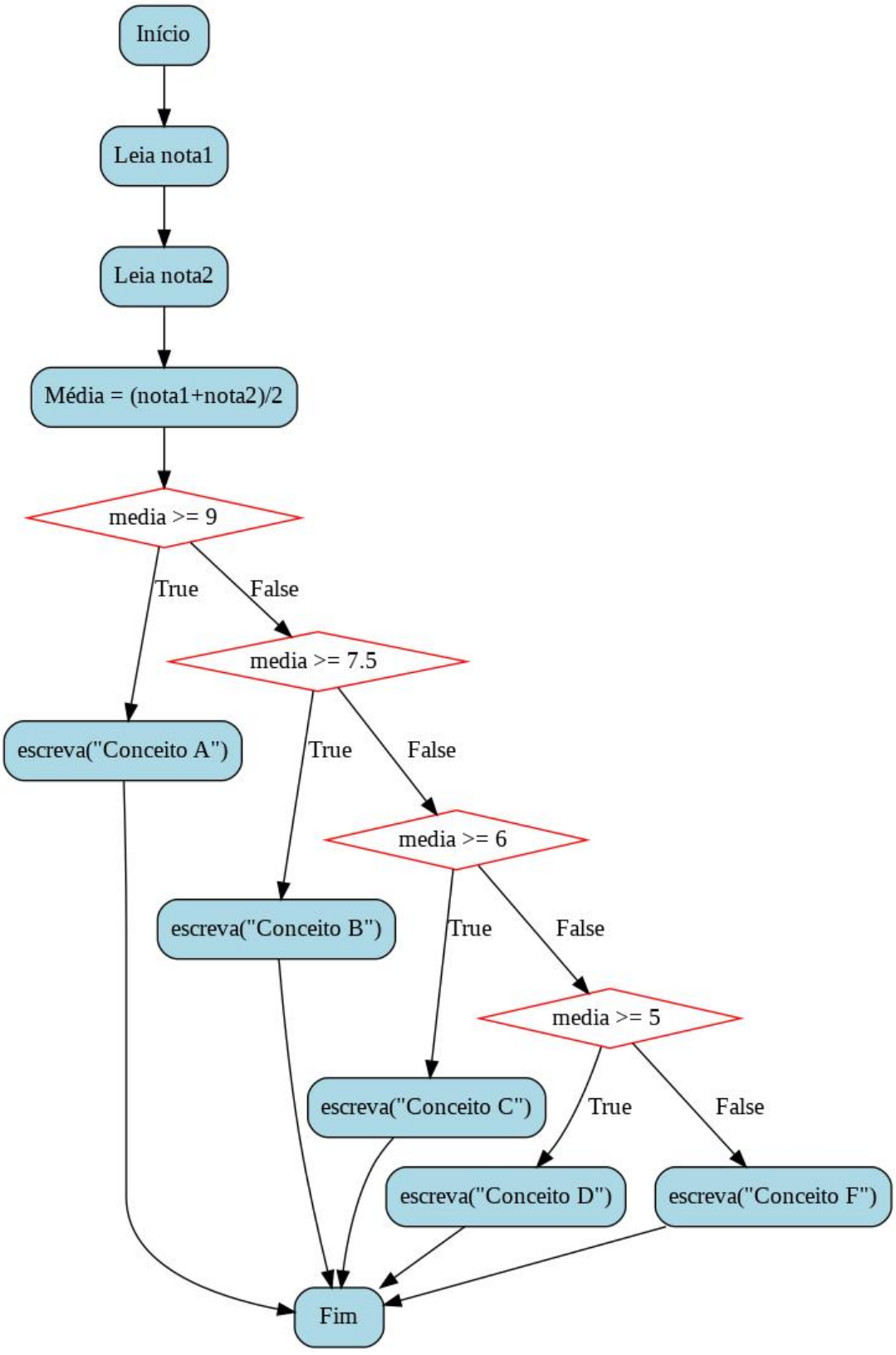
[ ]:
```
%%shell
gcc -Wall -std=c99 cap3ex01.c -o output2 -lm
./output2
# A biblioteca matemática deve ser vinculada ao construir o executável.
# Como fazer isso varia de acordo com o ambiente,
# mas no Linux / Unix, basta adicionar -lm ao comando
```

Exemplo 02 - Uso de Condicionais Compostas e Encadeadas

- Considere o seguinte pseudocódigo para ler duas notas reais, calcular a média das duas notas e atribui um conceito:

```
nota1 = leia("Digite a 1a nota:");
nota2 = leia("Digite a 2a nota:");
media = (nota1 + nota2)/2;
se media >= 9 então
    escreva("Conceito A");
senão se media >= 7.5
    escreva("Conceito B");
senão se media >= 6
    escreva("Conceito C");
senão se media >= 5|
    escreva("Conceito D");
senão
    escreva("Reprovado! Conceito F");
```

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



[ ]:
```
from txtoflow import txtoflow
```

```
txtoflow.generate(
    '''
    Início;
    Leia nota1;
    Leia nota2;
    Média = (nota1+nota2)/2;
    if ( media >= 9 ) {
        escreva("Conceito A");
    } else if (media >= 7.5 ) {
        escreva("Conceito B");
    } else if (media >= 6 ) {
        escreva("Conceito C");
    } else if (media >= 5 ) {
        escreva("Conceito D");
    } else {
        escreva("Conceito F");
    }
    Fim;
    '''
)
```

Casos para Teste Moodle+VPL

Para o professor criar uma atividade VPL no Moodle para este Exemplo 02, basta incluir em `Casos para teste`, o seguinte texto (pode incluir mais casos):

```
case=caso1
input=4.0
6.0
output=
Conceito D
case=caso2
input=4.0
5.0
output=
Conceito F
case=caso2
input=5.0
7.0
output=
Conceito C
case=caso3
input=6.0
9.0
output=
Conceito B
case=caso4
input=9.0
10.0
```

```
output=
Conceito A
```

[ ]:
```
%%writefile cap3ex02.c
#include <stdio.h>
float leia() {
  float valor;
  printf("Entre com um valor: ");
  scanf("%f", &valor);
  return valor;
}

int main(void) {

  // ENTRADAS
  float nota1 = leia();
  float nota2 = leia();

  // PROCESSAMENTO E SAÍDA
  float media = (nota1 + nota2)/2;
  if (media >= 9.0)
    printf("Conceito A");
  else if (media >= 7.5)
    printf("Conceito B");
  else if (media >= 6.0)
    printf("Conceito C");
  else if (media >= 5.0)
    printf("Conceito D");
  else
    printf("Reprovado! Conceito F.");
  return 0;
}
```

[ ]:
```
%%shell
gcc -Wall -std=c99 cap3ex02.c -o output2
./output2
```

1.6 Comando `switch`

O `switch` verifica se uma variável (do tipo `int` ou `char`) é ou não igual a certo valor constante (valor1, valor2, ..., valorN na sintaxe a seguir):

```
switch (variável) {
  case valor1:
    Comandos;
    break;
  case valor2:
    Comandos;
    break;
```

```
  case valorN:
    Comandos;
    break;
  default: // opcional, caso não ocorram os casos anteriores
    Comandos;
}
```

Esse comanto `switch` é útil quando se tem um menu de opções a ser executado.

Exemplo 03 - Exemplo de `switch` com menu de opções

[1]:
```
%%writefile cap3ex03.c
#include <stdio.h>
int main() {
  float area;
  int num;
  char escolha;
  printf("1. Circulo\n");
  printf("2. Quadrado\n");
  printf("Escolha:\n");

  scanf("%c", &escolha);

  switch (escolha) {
  case '1':
    printf("Raio:\n");
    scanf("%d", &num);
    area = 3.14 * num * num;
    printf("Area do circulo: ");
    printf("%.2f\n", area);
    break;

  case '2':
    printf("Lado:\n");
    scanf("%d", &num);
    area = num * num;
    printf("Area do quadrado: ");
    printf("%.2f\n", area);
    break;

  default:
    printf("Escolha Incorreta!\n");
  }
  return 0;
}
```

[3]:
```
%%shell
gcc -Wall -std=c99 cap3ex03.c -o output3
./output3
```

1.7 Exercícios

Ver notebook Colab no arquivo `cap3.part2.lab.*.ipynb` (`*` é a extensão da linguagem), utilizando aluma linguagem de programação de sua preferência, onganizadas em subpastas contidas de `"gen"`, na pasta do Google Drive colabs.

1.8 Revisão deste capítulo de Desvios Condicionais

- O que é um desvio condicional?
- Condições com lógica booleana
- Desvios condicionais Simples e Compostos, com Encadeamentos
- Exercícios
- Revisão deste capítulo de Desvios Condicionais