Algoritmos e Programação I: Aula 07*

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
http://ava.ufms.br

Sumário

1 Resolvendo Problemas com Estrutura Sequencial - (em elaboração)

Nesta aula daremos continuidade na apresentação de exemplos de problemas que utilizam na sua solução a estrutura de controle sequencial. Os programas (e algoritmos) solucionam problemas envolvendo cálculos, texto e gráficos. Alguns dos exemplos tratados também podem servir de inspiração na solução de exercícios da Lista 2.1. As soluções dos problemas seguem a metodologia descrita em sala:

a. Fase 1 - Descrição Detalhada (entendimento do problema).

b. Fase 2 - Domínio e Imagem do problema (Especificações de entrada e saída).

c. Fase 3 - Algoritmo (Refinamentos Sucessivos - Passos do Algoritmo).

d. Fase 4 - Codificação e Documentação (Finalização).

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina.

e. Fase 5 - Teste e Verificação do Programa.

Nesse próximos exemplos, faremos uma descrição mais resumida de cada uma das fases no desenvolvimento da solução dos próximos problemas. Considerando que os problemas que abordaremos são bem simples e que geralmente a entrada e saída estão bem definidas, temos que olhar com mais detalhe o projeto do algoritmo e os detalhes da codificação.

2 Resolvendo Problemas com Dados Numéricos

Vamos analisar problemas que usam a soma de valores dentro de um dado intervalo, e um problema que usa várias operações aritméticas para calcular medidas estatísticas. Esse último exemplo ilustra as limitações da estrutura sequencial e das variáveis simples, uma vez que as operações são feitas repetidamente com as variáveis de entrada e o conceito de variável simples dificulta o número de amostras que podem ser analisadas pelas medidas estatísticas.

2.1 Somas de Medidas de Tempo

Dados duas unidades de tempo dadas em `horas`, `minutos` e `segundos`, onde minutos e segundos são valores no intervalo $[0, 59]$, projetar um algoritmo para computar a soma entre essas duas unidades de tempo, onde a soma dos segundos e minutos devem ser valores no intervalo $[0, 59]$.

Vamos projetar um algoritmos chamado `TempoDecorrido` para computar a soma de duas medidas de tempo dadas em horas, minutos e segundos. O exemplo abaixo ilustra uma entrada e a saída desejada na solução do problema.

Exemplo:

```
# formato da entrada
2 15 30 # entrada da primeira medida de tempo
1 25 41 # entrada da segunda medida de tempo

# formato da saída
 2 horas 15 minutos 30 segundos +
 1 horas 25 minutos 41 segundos =
 3 horas 41 minutos 11 segundos
```

Temos que a descrição nesse caso é praticamente a própria definição do problema.

```
# Este algoritmo lê duas medidas de tempo em horas, minutos e  segundos,
# onde onde minutos e segundos são valores no intervalo [0, 59] e computa
# a soma das medidas em horas, minutos e segundos, de tal forma que
# os valores de minutos e segundos permaneçam no intervalo [0, 59] e
# imprima o resultado
```

As Especificações da entrada e saída (domínio e imagem):

Entrada	Saída

Horas1, Minutos1 e Segundos1
Horas2, Minutos2 e Segundos2
`horas1`, `minutos1`, `segundos1`
`horas2`, `minutos2`, `segundos2`
	Total Horas, Total Minutos, Total Segundos
	`totsegundos`, `totminutos`, `tothoras`

Nesse exemplo as variáveis são compostas de tipos básicos. Necessitamos de `int` para armazenar os valores de `segundos1`, `segundos2`, `minutos1`, `minutos2`, `horas1`, `horas2`, `totsegundos`, `totminutos` e `tothoras`. Os identificadores abaixo representam as variáveis do tipo inteiro para armazenar as informações da entrada e saída:

```
# descrição das variáveis utilizadas
# int horas1, minutos1, segundos1, horas2, minutos2, segundos2
# int tothoras, totminutos, totsegundos
```

Quando da implementação do algoritmo pode surgir a necessidade de utilizarmos variáveis auxiliares para armazenar valores intermediários.
Exemplo:

```
# formato da entrada
2 15 30 # entrada da primeira medida de tempo
1 25 41 # entrada da segunda medida de tempo

# estado das variáveis após a leitura
segundos1 == 30
minutos1 == 15
horas1 == 2
segundos2 == 41
minutos2 == 25
horas2 == 1

# cálculo da soma dos segundos no intervalo [0,59]
(30 + 41) % 60 = 11
# cálculo da soma dos minutos no intervalo [0, 59]
(30 + 41) // 60 = 1 # minutos remanescentes da soma de segundos
((15 + 25 + (30 + 41) // 60) % 60 = 41
# cálculo da soma das horas
((15 + 25 + (30 + 41) // 60) // 60 = 0 # horas reman. da soma de minutos
2 + 1 + ((15 + 25 + (30 + 41) // 60) // 60 = 3

# estado das variáveis após a soma
totsegundos == 11
totminutos == 41
tothoras == 3
```

As pré e pós-condições ficam:

```
# pré: horas1 minutos1 segundos1 horas2 minutos2 segundos2
```

```
# pós: tothoras == (horas1 + horas2 + (minutos1 + minutos2 + (segundos1 +
#      segundos2)//60)//60 and totminutos == (minutos1 + minutos2 +
#      (segundos1 + segundos2)%60 and totsegundos == (segundos1 +
#      segundos2)%60
```

Os passos do algoritmo podem ser descritos como:

```
Algoritmo: Elapsed
# Passo 1. Leia as medidas em horas, minutos e segundos
# Passo 1.1. Leia a medidas 1 em horas, minutos e segundos
# Passo 1.2. Leia a medidas 2 em horas, minutos e segundos
# Passo 2. Calcule a soma das medidas
# Passo 2.1. Calcule o total de segundos
# Passo 2.2. Calcule o total de minutos
# Passo 2.3. Calcule o total de horas
# Passo 3. Imprima a somas das medidas
# fim do Algoritmo
```

Após a descrição dos passos do algoritmo, passamos para a fase da finalização da codificação e documentação na linguagem Python.

```
1  # -*- coding: utf-8 -*-
2  # Programa: elapsed01.py
3  # Programador:
4  # Data: 22/09/2010
5  # Este algoritmo lê duas medidas de tempo em horas, minutos e  segundos,
6  # onde onde minutos e segundos são valores no intervalo [0, 59] e computa
7  # a soma das medidas em horas, minutos e segundos, de tal forma que
8  # os valores de minutos e segundos permaneçam no intervalo [0, 59] e
9  # imprima o resultado
10 # início do módulo principal
11 # descrição das variáveis utilizadas
12 # int horas1, minutos1, segundos1, horas2, minutos2, segundos2
13 # int minutos, horas, tothoras, totminutos, totsegundos
14
15 # pré: horas1 minutos1 segundos1 horas2 minutos2 segundos2
16
17 # Passo 1. Leia as medidas em horas, minutos e segundos
18 # Passo 1.1. Leia a medidas 1 em horas, minutos e segundos
19 print('Entre com as horas, minutos e segundos da primeira leitura: ')
20 horas1, minutos1, segundos1 = map(int, input().split())
21 # Passo 1.2. Leia a medidas 2 em horas, minutos e segundos
22 print('Entre com as horas, minutos e segundos da segunda leitura: ')
23 horas2, minutos2, segundos2 = map(int, input().split())
24 # Passo 2. Calcule a soma das medidas
25 # Passo 2.1. Calcule o total de segundos
26 totsegundos = (segundos1 + segundos2) % 60
27 # Passo 2.2. Calcule o total de minutos
28 minutos = (segundos1 + segundos2) // 60
29 totminutos = ((minutos1+minutos2) + minutos)%60
```

```
# Passo 2.3. Calcule o total de horas
horas = (minutos1+minutos2 + minutos)//60
tothoras = horas1 + horas2 + horas
# Passo 3. Imprima a somas das medidas
print('{0:2d} horas {1:2d} minutos {2:2d} segundos +'.format(horas1,\
                                                    minutos1, segundos1))
print('{0:2d} horas {1:2d} minutos {2:2d} segundos ='.format(horas2, \
                                                    minutos2, segundos2))
print('{0:2d} horas {1:2d} minutos {2:2d} segundos'.format(tothoras, \
                                                totminutos, totsegundos))

# pós: tothoras == (horas1 + horas2 + (minutos1 + minutos2 + (segundos1 +
#      segundos2)//60)//60 and totminutos == (minutos1 + minutos2 +
#      (segundos1 + segundos2)%60 and totsegundos == (segundos1 +
#      segundos2)%60
# fim do módulo principal
```

Na fase da verificação e teste, projete um conjunto de testes que explore todas as possibilidades do programa.
Exemplo:

```
# formato da entrada
Entre com as horas, minutos e segundos da primeira leitura: 2 15 30
Entre com as horas, minutos e segundos da segunda leitura: 1 25 41

# formato da saída
 2 horas 15 minutos 30 segundos +
 1 horas 25 minutos 41 segundos =
 3 horas 41 minutos 11 segundos
```

2.2 Variância

Em Estatística, várias medidas auxiliam o entendimento das amostras. Entre essas medidas, a média, variância e desvio padrão desempenham um papel muito importante. Vamos analisar o problema de obter a média, variância e o desvio padrão de uma conjunto de 4 notas. A média e a variância dos números $x_1, x_2, \cdots, x_n$ podem ser calculadas usando as fórmulas:

$$média = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$variância = \frac{1}{n}\sum_{i=1}^{n} {x_i}^2 - \frac{1}{n^2}(\sum_{i=1}^{n} x_i)^2$$

e o desvio padrão é a raiz quadrada da variância.
Exemplo:

```
# formato da entrada
7.0 5.5 6.0 5.8 # entrada das notas
```

```
# formato da saída
Primeira Prova     7.0
Segunda Prova      5.5
Terceira Prova     6.0
Quarta Prova       5.8
Média das Provas  6.08
Variância         0.32
Desvio Padrão     0.56
```

A descrição acrescenta mais alguns detalhes na formulação do problema.

```
# Este programa lê quatro notas de provas variando de 0.0 a10.0,
# (o usuário entra com as notas dentro dos limites), calcula a média a
# variância e o desvio padrão das quatro notas das provas. Os resultados
# são impressos usando um formato dado de saída.
```

As especificações de entrada e saída (domínio e imagem) ficam:

Entrada	**Saída**
Notas das quatro provas [0.0,10.0] `prova1`, `prova2`, `prova3` e `prova4`	Média, Variância e Desvio Padrão das Notas `mediaprovas`, `desviopadrao`, e `variancia`

Os identificadores abaixo representam as variáveis do tipo `float` para armazenar as informações da entrada e saída:

```
# descrição das variáveis utilizadas
# float prova1, prova2, prova3, prova4
# float mediaprovas, desviopadrao, variancia
```

Exemplo:

```
# formato da entrada
7.0 5.5 6.0 5.8 # entrada das notas

# estado das variáveis após a leitura
prova1 == 7.0
prova2 == 5.5
prova3 == 6.0
prova4 == 5.8

# cálculo da média
somaprovas = prova1 + prova2 + prova3 + prova4 # usaremos somaprovas
somaprovas = 7.0 + 5.5 + 6.0 + 5.8
mediaprovas = somaprovas/4.0
mediaprovas = 24.3/4.0
# cálculo do valor da variância
somaquadrado = prova1**2 + prova2**2 + prova3**2 + prova4**2
somaquadrado = 7.0**2 + 5.5**2 + 6.0**2 + 5.8**2
```

```
variancia = somaquadrado/4.0 - (somaprovas**2)/(4.0**2)
variancia = 148.89/4.0 - 590.49/16.0
# cálculo do desviopadrao
desviopadrao = sqrt(variancia)
desviopadrao = sqrt(0.316875)

# estado das variáveis após o cálculo da média, variância e desvio padrão
somaprovas == 24.3
mediaprovas == 6.075
somaquadrado == 148.89
variancia == 0.316875
desviopadrao == 0.56291651
```

Em função das operações, necessitamos de mais duas variáveis para armazenar os valores intermediários:

```
# descrição das variáveis utilizadas
# float prova1, prova2, prova3, prova4
# float somaprovas, mediaprovas, somaquadrados
# float desviopadrao, variancia
```

Considerando a descrição do problema e as especificações da entrada e saída, as pré e pós-condições ficam:

```
# pré: prova1, prova2, prova3, prova4

# pós: media == (Soma i em {1,2,3,4}: prova[i])/4.0 and
#      variancia == ((1/n)*(Soma i em {1,2,3,4}: (prova[i])**2) -
#      (1/n**2)*(Soma i em {1,2,3,4}: prova[i])**2)) and
#      desviopadrao == sqrt(variancia)
```

Vamos agora descrever os passos do algoritmo que implementa uma solução para esse problema. Numa primeira análise do problema temos:

```
Algoritmo: MédiaVariância
# Passo 1. Leia as notas
# Passo 2. Calcule a média, variância e desvio padrão
# Passo 3. Imprima a média, a variância e o desvio padrão
# fim do Algoritmo
```

O Passo 2 necessita de um refinamento, pois precisamos detalhar mais esse passo para que ele possa ser implementado num computador:

```
# Passo 2. Calcule a média, variância e desvio padrão
# Passo 2.1. Calcule a média das provas
# Passo 2.2. Calcule a variância
# Passo 2.3. Calcule o desvio padrão
```

No caso do Passo 2.1, podemos calcular a soma das notas e dividir por 4.0, fazer o cálculo diretamente. Como vimos acima, o valor da soma das notas será utilizado no decorrer dos outros cálculos. No cálculo da variância, necessitamos efetuar a soma dos quadrados das notas. Em função disso, temos também que fazer um refinamento do Passo 2.2. Uma vez computada a variância, o desvio padrão pode ser obtido usando a raiz quadrada da variância. Com isso, o Passo 2 fica:

```
# Passo 2. Calcule a média, variância e desvio padrão
# Passo 2.1. Calcule a soma das notas das provas
# Passo 2.2. Calcule a média das provas
# Passo 2.3. Calcule a soma dos quadrados das notas provas
# Passo 2.4. Calcule a variância
# Passo 2.5. Calcule o desvio padrão
```

O Passo 3 também precisa de um detalhamento, mas como se trata de impressão, faremos isso diretamente na implementação do algoritmo em Python. Para obter o valor da raiz quadrada de um número `float` usaremos a função/método `math.sqrt(x)` da linguagem *Python*. A codificação em Python fica:

```
1  # -*- coding: utf-8 -*-
2  # Programa: variancia.py
3  # Programador:
4  # Data: 23/08/2016
5  # Este programa lê quatro notas de provas variando de 0.0 a10.0,
6  # (o usuário entra com as notas dentro dos limites), calcula a média a
7  # variância e o desvio padrão das quatro notas das provas. Os resultados
8  # são impressos usando um formato dado de saída.
9  # declaração dos módulos utilizados
10 import math
11 # início do módulo principal
12 # descrição das variáveis utilizadas
13 # float prova1, prova2, prova3, prova4
14 # float somaprovas, mediaprovas, somaquadrados
15 # float desviopadrao, variancia
16
17 # pré: prova1, prova2, prova3, prova4
18
19 # Passo 1. Leia as notas
20 prova1, prova2, prova3, prova4 = map(float, input().split())
21 # Passo 2. Calcule a média, variância e desvio padrão
22 # Passo 2.1. Calcule a soma das notas das provas
23 somaprovas = prova1 + prova2 + prova3 + prova4
24 # Passo 2.2. Calcule a média das provas
25 mediaprovas = somaprovas/4.0
26 # Passo 2.3. Calcule a soma dos quadrados das notas provas
27 somaquadrado  = prova1**2 + prova2**2 + prova3**2 + prova4**2
28 # Passo 2.4. Calcule a variância
29 variancia = somaquadrado/4.0 - (somaprovas**2)/(4.0**2)
30 # Passo 2.5. Calcule o desvio padrão
31 desviopadrao = math.sqrt(variancia)
```

```
# Passo 3. Imprima a média, a variância e o desvio padrão
print('Primeira Prova    {0:4.1f}'.format(prova1))
print('Segunda Prova     {0:4.1f}'.format(prova2))
print('Terceira Prova    {0:4.1f}'.format(prova3))
print('Quarta Prova      {0:4.1f}'.format(prova4))
print('Média das Provas {0:5.2f}'.format(mediaprovas))
print('Variância        {0:5.2f}'.format(variancia))
print('Desvio Padrão    {0:5.2f}'.format(desviopadrao))

# pós: media == (Soma i em {1,2,3,4}: prova[i])/4.0 and
#      variancia == ((1/n)*(Soma i em {1,2,3,4}: (prova[i])**2) -
#      (1/n**2)*(Soma i em {1,2,3,4}: prova[i])**2)) and
#      desviopadrao == sqrt(variancia)
# fim do módulo principal
```

Exemplo 1:

```
# formato da entrada
7.0 5.5 6.0 5.8

# formato da saída
Primeira Prova     7.0
Segunda Prova      5.5
Terceira Prova     6.0
Quarta Prova       5.8
Média das Provas  6.08
Variância         0.32
Desvio Padrão     0.56
```

Exemplo 2:

```
# formato da entrada
7.0 7.0 7.0 7.0

# formato da saída
Primeira Prova     7.0
Segunda Prova      7.0
Terceira Prova     7.0
Quarta Prova       7.0
Média das Provas  7.0
Variância         0.00
Desvio Padrão     0.00
```

Implemente e execute o programa para vários conjuntos de notas. Como fica o algoritmo/programa se forem 5 notas? E se forem 100 notas?

3 Resolvendo Problemas com Texto

O Python tem uma série de recursos poderosos para manipular strings. Na medida em que formos necessitando, descreveremos como esses recursos podem ser utilizados na solução de problemas computacionais usando strings. Um desses recursos é o *slicing*. O *slicing* permite "fatiarmos" uma string em pedaços, inverter strings, analisar sufixos e prefixos, etc. Dado uma string `palavra`, `palavra[a:b]` corresponde aos caracteres `palavra[a]..palavra[b-1]`.
Exemplo:

```
palavra = 'Algoritmos'
parte = palavra[2:6]
# parte = palavra[2]..palavra[5]
# estado da variável parte
parte == 'gori'
```

além disso, podemos usar índices negativos e nesse caso o deslocamento será da direita para esquerda. Temos que `palavra[-1]` corresponde ao último caractere de `palavra`, `palavra[tam-1]`, onde `tam = len(palavra)`. Um outro recurso muito usado é a inversão dos caracteres da string usando `[:-1]`. Nesse caso, os caracteres da string são acessados da direita para esquerda.
Exemplo:

```
palavra = 'Algoritmos'
# palavra[-2] == palavra[8]
caractere = palavra[-2]
# estado da variável caractere
caractere == 'o'
inverte = palavra[:-1]
# estado da variável inverte
inverte == 'somtiroglA'
```

Vamos agora descrever uma forma de como codificar uma palavra. Como só dispomos da estrutura sequencial, fica muito "tedioso" codificar palavras com muitos caracteres. Na medida que novas estruturas de controle forem sendo disponibilizadas, revisitaremos esse problema.

3.1 Cifrador de César I

Vimos anteriormente que o *Cifrador de César* é um esquema de codificação na qual cada letra é substituída por uma letra diferente do alfabeto, digamos a quinta letra seguindo-a. Desta forma a mensagem

```
algoritmos
```

é codificada no Cifrador de César usando o deslocamento igual a 5 como

```
fqltwnyrtx
```

Como só temos ainda a estrutura sequencial para projetar algoritmos, só conseguiremos codificar eficientemente uma palavra com um número pequeno de caracteres. Vamos descrever uma solução para codificar um palavra de cinco (5) caracteres maiúsculos usando o cifrador de César.

Na codificação fazemos uma associação numérica a cada um dos caracteres de `'A'` (0) a `'Z'` (25). A cada um dos caracteres é somado o valor do deslocamento de tal forma que o valor da soma fique no intervalo $[0, 25]$. Uma forma de resolver esse problema é usar as funções que convertem um caractere para um número inteiro correspondente ao código ASCII (`ord(x)`) e um número inteiro, código ASCII, para caractere (`chr()`). O intervalo dos valores decimais da tabela ASCII dos caracteres maiúsculos é {`65 = 'A', 66 = 'B', ..., 90 = 'Z'`}. Em função disso, temos que converter esses valores para o intervalo $[0, 25]$, somar o deslocamento, manter o valor da soma no intervalo $[0, 25]$, converter o valor no intervalo $[65, 90]$ e finalmente associar o caractere correspondente.

```
letra = 'M'
codigo = ord(letra)
codigo = ord('M')

# estado da variável após a atribuição
codigo  == 77

# converter o valor para o intervalo [0,25]
codigo = ord('M') - ord('A')
codigo = 77 - 65
codigo = 12

# somar o deslocamento
deslocamento = 5
codigo = codigo + deslocamento
codigo = 12 + 5
codigo = 17

# manter no intervalo [0,25]
codigo = codigo % 26

# converter o valor para o intervalo [65,90]
codigo  = codigo + ord('A')
codigo = 17 + 65
codigo = 82

# computar o caractere correspondente
letraC = chr(codigo)
letraC = chr(82)
letraC = 'R'
```

Vamos agora descrever uma forma de como codificar uma dada palavra com 5 caracteres usando o Codificador de César. Dada uma palavra com 5 caracteres do alfabeto {`'A'..'Z'`} e um número inteiro entre 0 e 25, codificaremos a palavra convertendo cada caractere da palavra para o seu código numérico ASCII correspondente e adicionando um inteiro positivo

entre 0 e 25. O valor numérico obtido nessa operação será novamente convertido para o caractere correspondente do código ASCII. Consideramos o alfabeto de forma circular, ou seja, `'Z' + 1 == 'A'`. Posteriormente, na medida em que formos aprendendo como manipular outros tipos de estruturas para solucionar problemas usando um computador, poderemos ampliar o escopo da entrada para o cifrador de César.

Exemplo:

```
# formato da entrada
AULAS # entrada da string
5 # valor do deslocamento

# formato da saída
FZQFX
```

A entrada é dada por uma string com 5 caracteres ASCII pertencentes ao alfabeto `'A'-'Z'` (a palavra da entrada só é composta por caracteres maiúsculos do alfabeto), e um inteiro no intervalo $[0, 25]$.

A descrição detalha mais algumas características do problema e fornece uma melhor especificação para solucionar o problema.

```
# Este programa lê uma palavra composta por 5 caracteres maiúsculos do
# alfabeto {'A'-'Z'} e um número inteiro no intervalo [0,25]. O programa
# codifica a palavra usando o deslocamento dos caracteres da string de
# entrada e imprime a palavra codificada. O programa usa os códigos ASCII,
# e o Alfabeto é tratado de forma circular, ou seja, 'Z' + 1 == 'A'.
```

Com isso, as especificações de entrada e saída ficam:

Entrada	**Saída**
Uma palavra com 5 caracteres do alfabeto {'A'-'Z'} e um número inteiro no intervalo $[0, 25]$ `palavra`, `num`	Uma palavra codificada com o Cifrador de César usando o deslocamento dos caracteres `codificada`

Os identificadores abaixo representam as variáveis do tipo `str` para armazenar a palavra e sua codificação, e do tipo `int` para armazenar o deslocamento a ser dado nos caracteres da palavra.

```
# Descrição das variáveis utilizadas
# str palavra, codificada
# int deslocamento
```

Exemplo:

```
# formato da entrada
AULAS # entrada da string
5 # valor do deslocamento

# estado das variáveis após a leitura
```

```
palavra == 'AULAS'
deslocamento == 5

# codificação da palavra, caractere a caractere
letra = 'A'
# converter o valor para o intervalo [0,25]
codigo = ord('A') - ord('A')

# somar o deslocamento
codigo = codigo + deslocamento
codigo = 0 + 5
codigo = 5

# manter no intervalo [0,25]
codigo = codigo % 26

# converter o valor para o intervalo [65,90]
codigo  = 5 + ord('A')
codigo = 5 + 65
codigo = 70

# computar o caractere correspondente
letraC = chr(codigo)
letraC = chr(70)
letraC = 'F'

letra = 'U'
# converter o valor para o intervalo [0,25]
codigo = ord('U') - ord('A')

# somar o deslocamento
codigo = codigo + deslocamento
codigo = 20 + 5
codigo = 25

# manter no intervalo [0,25]
codigo = codigo % 26

# converter o valor para o intervalo [65,90]
codigo  = 25 + ord('A')
codigo = 25 + 65
codigo = 90

# computar o caractere correspondente
letraC = chr(codigo)
letraC = chr(90)
letraC = 'Z'

...
```

```
# juntando os passos
letraC = chr((ord(letra) - ord('A') + codificador)%26 + ord('A'))
```

Após um melhor entendimento do problema, temos que as pré e pós-condições ficam:

```
# pré: palavra and palavra[i] em {'A'..'Z'} deslocamento

# pós: codificada and codificada[i] em {'A'..'Z'} and
       para i em {0,...,4}:codificada[i]==palavra[i] + deslocamento
```

Nessas especificações, usamos a descrição dos caracteres de uma string. No decorrer da disciplina apresentaremos mais detalhes sobre strings. Vamos agora descrever os passos do algoritmo para a solução do problema.

```
Algoritmo: Cifrador de César
# Passo 1. Leia uma palavra com 5 caracteres maiúsculos e o deslocamento
# Passo 1.1. Leia uma palavra
# Passo 1.2. Leia o deslocamento
# Passo 2. Codifique a palavra
# Passo 3. Imprima a palavra codificada
# fim do Algoritmo
```

O Passo 2 necessita um refinamento. Como vimos acima, a codificação da palavra é feita caractere a caractere.

```
# Passo 2. Codifique a palavra
# Passo 2.1. Codifique o primeiro caractere
# Passo 2.2. Codifique o segundo caractere
# Passo 2.3. Codifique o terceiro caractere
# Passo 2.4. Codifique o quarto caractere
# Passo 2.6. Codifique o quinto caractere
# Passo 2.7. Concatene os caracteres
```

Uma vez finalizado os refinamentos sucessivos, vamos codificar a solução na linguagem Python. Os Passos 1 e 3 são facilmente implementados em Python. Para implementarmos o Passo 2 temos que entender como o Python acessa cada um dos caracteres da string `palavra`. Considerando que a string de entrada é composta por 5 caracteres, o acesso a cada um dos caracteres de `palavra` pode ser feito da seguinte maneira:

```
palavra = 'AULAS'
palavra[0] == 'A'
palavra[1] == 'U'
palavra[2] == 'L'
palavra[3] == 'A'
palavra[4] == 'S'
```

onde `palavra[i]`, $0 \leq i \leq 4$ armazena o i-ésimo caractere da string `palavra`.

Como vimos anteriormente, a codificação de um caractere pode ser feita com:

```
letraC = chr((ord(letra) - ord('A') + codificador)%26 + ord('A'))
```

Como veremos mais a frente, o tipo string é imutável, ou seja, não é possível atribuir ou alterar caracteres individuais no tipo string. O que podemos fazer é concatenar um novo caractere ao final da string, mas não é "possível" alterar um caractere existente numa string. Nesse exercício vamos codificar cada um dos cinco caracteres da string e ao final concatená-los para obter a palavra codificada.

```
# formato da entrada
AULAS
5

# estado das variáveis após a leitura
palavra == 'AULAS'
deslocamento == 5

# letraC = chr((ord(letra) - ord('A') + codificador)%26 + ord('A'))
# codifica os 5 caracteres da variável palavra
char0 = chr((ord(palavra[0]) - ord('A') + codificador)%26 + ord('A'))
char1 = chr((ord(palavra[1]) - ord('A') + codificador)%26 + ord('A'))
char2 = chr((ord(palavra[2]) - ord('A') + codificador)%26 + ord('A'))
char3 = chr((ord(palavra[3]) - ord('A') + codificador)%26 + ord('A'))
char4 = chr((ord(palavra[4]) - ord('A') + codificador)%26 + ord('A'))
# concatena os caracteres codificados
codificada = char0 + char1 + char2 + char3 + char4

# estado da variável após a codificação
codifica == 'FZQFX'
```

Em função da codificação de cada um dos caracteres da variável `palavra`, necessitamos de 5 novas variáveis para armazenar cada um dos caracteres. No Python, um caractere é considerado uma string. Com isso as variáveis necessárias para o programa são:

```
# descrição das variáveis utilizadas
# str palavra, codificada
# str char0, char1, char2, char3, char4
# int deslocamento
```

O programa `cifrador00.py` abaixo implementa o algoritmo do cifrador de César.

```
1  # -*- coding: utf-8 -*-
2  # Programa: codifica00.py
3  # Programador:
4  # Data: 27/04/2016
5  # Este programa lê uma palavra composta por 5 caracteres maiúsculos do
6  # alfabeto {'A'-'Z'} e um número inteiro no intervalo [0,25]. O programa
7  # codifica a palavra usando o deslocamento dos caracteres da string de
```

```
# entrada e imprime a palavra codificada. O programa usa os códigos ASCII,
# e o Alfabeto é tratado de forma circular, ou seja, 'Z' + 1 == 'A'.
# início do módulo principal
# descrição das variáveis utilizadas
# str palavra, codificada
# str char0, char1, char2, char3, char4
# int deslocamento

# pré: palavra and palavra[i] em {A..Z} deslocamento

# Passo 1. Leia uma palavra com 5 caracteres maiúsculos e o deslocamento
# Passo 1.1. Leia uma palavra de 5 caracteres maiúsculos do alfabeto
palavra = input('Entre com uma palavra com 5 caracteres maiúsculos: ')
# Passo 1.2. Entre com o deslocamento a ser dado para os caracteres.
codificador = int(input('Entre com um inteiro ente 0 e 25: '))
# Passo 2. Codifique os caracteres da palavra
# Passo 2.1. Codifique o primeiro caractere
char0 = chr((ord(palavra[0]) - ord('A') + codificador)%26 + ord('A'))
# Passo 2.2. Codifique o segundo caractere
char1 = chr((ord(palavra[1]) - ord('A') + codificador)%26 + ord('A'))
# Passo 2.3. Codifique o terceiro caractere
char2 = chr((ord(palavra[2]) - ord('A') + codificador)%26 + ord('A'))
# Passo 2.4. Codifique o quarto caractere
char3 = chr((ord(palavra[3]) - ord('A') + codificador)%26 + ord('A'))
# Passo 2.6. Codifique o quinto caractere
char4 = chr((ord(palavra[4]) - ord('A') + codificador)%26 + ord('A'))
# Passo 2.7. Concatene os caracteres
codificada = char0 + char1 + char2 + char3 + char4
# Passo 3. Imprima a mensagem codificada
print('Codificada: {0:s}'.format(codificada))

pós: codificada and codificada[i] em {A..Z} and
     para i em {0,...,4}:codificada[i]==palavra[i] + deslocamento
# fim do módulo principal
```

Como vimos acima, a função `ord(caractere)` converte um caractere para o seu respectivo código ASCII, e a função `chr(decimal)` converte um número decimal para o seu respectivo código ASCII. Além disso, relembramos que a concatenação de duas ou mais palavras é feita com o operador ‘‘`+`’’, que foi usado para atribuir os caracteres codificados para a variável `codificada`. Posteriormente veremos mais detalhes da manipulação do tipo string no Python.

Observe que na codificação de cada caractere da palavra subtraímos o valor decimal do caractere `'A'`, para obter um valor no intervalo $[0, 25]$.

```
chr((ord(palavra[0]) - ord('A') + codificador)%26 +  ord('A'))
```

e adicionamos o valor do codificador, um número entre 0 e 26. Para obtermos o caractere correspondente no alfabeto, temos que garantir que o valor da soma fique em entre 0 (para o caractere `'A'` e 25 (para o caractere `'Z'`). Isso é feito computando a soma módulo 26. Depois temos que somar o valor correspondente ao caractere `ord('A')` para obtermos o valor

decimal do caractere correspondente na tabela ASCII. Usando a função `chr()` obtemos o caractere correspondente. Após editar o programa, usando o interpretador Python, verifique se o programa não possui nenhum erro de sintaxe e execute-o para os exemplos abaixo:

Exemplo 1

```
# formato da entrada
AAAAA
2

# formato da saída
CCCCC
```

Exemplo 2

```
# formato da entrada
ZZZZZ
4

# formato da saída
DDDDD
```

Exemplo 3

```
# formato da entrada
AULAS
5

# formato da saída
FZQFX
```

(a) Teste o programa para várias palavras com 5 caracteres maiúsculos.

(b) O que ocorre se a sua mensagem tem mais de cinco caracteres?

(c) O que ocorre quando a palavra tem menos de 5 caracteres?

(d) O que ocorre quando a palavra tem 5 caracteres, mas algum caractere que não seja um caractere entre {'A'..'Z'}?

4 Exercícios

4.1 Média

Escreva um programa em Python para calcular a média final de uma dada disciplina. A disciplina é avaliada por meio de três provas e dois trabalhos. No cálculo da média final, as provas tem peso 3 e os trabalhos tem peso 1. A entrada é dada por 5 números reais, onde cada número indica uma nota do estudante (três notas de prova e duas de trabalhos). As três notas das provas e as duas notas dos trabalhos são fornecidas em 2 linhas, uma

linha para as três notas das provas e outra linha para as duas notas dos trabalhos. A saída consiste em imprimir a média final da disciplina usando o formato de saída descrito abaixo. O valor da média final deve ser impresso com 4 espaços, sendo uma casa decimal. O (.) também ocupa um espaço e para obter esse formato dê uma lida no material sobre o `print` ({0:4.1f}) para o Python.

Exemplo 1

```
# formato da entrada
7.0 3.5 6.0
4.0 9.0


# formato da saída
Média Final:  5.7
```

Exemplo 2

```
# formato da entrada
7.0 7.0 7.0
7.0 7.0

# formato da saída
Média Final:  7.0
```

4.2 Total de Segundos II

Projete e implemente um programa em `Python` para ler uma medida de tempo dada por uma quádrupla de números inteiros, cada quádrupla indicando uma quantidade de tempo expressa em dias, horas, minutos e segundos, e compute e imprima o total de segundos dessa medida de tempo. A entrada do seu programa deve ser dada por uma quádrupla de números inteiros, onde cada quádrupla indica o número de dias, horas ($0 \leq$ horas < 24), minutos ($0 \leq$ minutos < 60) e segundos ($0 \leq$ segundos < 60) de uma dada medida de tempo. A saída é o total em segundos da medida de tempo lida impressa no formato especificado nos exemplos abaixo.

Exemplo 1

```
# formato da entrada
1 3 34 45

# formato da saída
1 Dias + 3 Horas + 34 Minutos + 45 Segundos = 99285 Segundos
```

Exemplo 2

```
# formato da entrada
10 5 19 25

# formato da saída
```

```
10 Dias + 5 Horas + 19 Minutos + 25 Segundos = 883165 Segundos
```

4.3 Tempo Decorrido II

Dado uma medida de tempo em segundos, computar e imprimir o equivalente em dias, horas, minutos e segundos, onde horas < 24, minutos < 60 e segundos < 60. Nesse problema usamos uma abordagem que é utilizada em vários outros problemas. Os resultados são obtidos pela computação repedida de resto (%) e divisão inteira (//). Se dividirmos o total em segundos por 60, o quociente dá o total de minutos e o resto o total de segundos que sobra. Analogamente, dividimos o total de minutos por 60. O resto é a quantidade de minutos e o quociente o número de horas. Repetimos o procedimento uma vez mais, dividindo o número de horas por 24. O resto será o número de horas e o quociente o número de dias. Para um valor de 90 segundos, a resultado esperado é de 1 minuto (`90 // 60`) e 30 (`90 % 60`) segundos.

Exemplo 1

```
# formato da entrada
90 # total de segundos

# formato da saída
0 0 1 30
```

Exemplo 2:

```
# formato da entrada
95410 # entrada em segundos

# formato da saída
1 2 30 10
```

```
Algoritmo: TempoDecorrido II
# Este algoritmo lê uma medida de tempo em segundos, calcula o valor
# equivalente horas, minutos e segundos e imprime o resultado.
# Passo 1. Leia o total de segundos
# Passo 2. Calcule o total de horas, minutos e segundos
# Passo 2.1. Calcule o Total de Segundos
# Passo 2.2. Calcule o total de minutos
# Passo 2.3. Calcule o total de horas
# Passo 2.4. Calcule o total de dias
# Passo 3. Imprima o resultado
# fim do Algoritmo
```

Leia o material das aulas e implemente uma solução para o problema do tempo decorrido em dias, horas, minutos e segundos de acordo com as especificações acima. Se você editou o programa corretamente, a execução do comando de interpretação não gerará nenhuma advertência ou erro. Caso isso ocorra algum erro ou advertência, veja o número da linha e verifique o que você digitou de forma errada. Após ter corrigido as possíveis advertências e erros, execute o programa.

4.4 Cifrador de César

Descrevemos acima um algoritmo que codifica uma palavra com 5 caracteres. Usando esse exemplo, projete e implemente um programas em Python que leia uma palavra de 6 caracteres e um número inteiro positivo e codifique a mensagem usando o Cifrador de César e imprima a mensagem codificada. Para esse exercício a palavra a ser codificada será dada por um conjunto de caracteres do alfabeto, sem o caractere espaço, números, ou quaisquer sinais de pontuação ou caracteres especiais. Para simplificar, a mensagem a ser codificada será toda expressa em caracteres maiúsculos.

```
Algoritmo: TempoDecorrido II
# -*- coding: utf-8 -*-
# Programa: codifica00.py
# Programador:
# Data: 27/04/2016
# Este programa lê uma palavra composta por 5 caracteres maiúsculos do
# alfabeto {'A'-'Z'} e um número inteiro no intervalo [0,25]. O programa
# codifica a palavra usando o deslocamento dos caracteres da string de
# entrada e imprime a palavra codificada. O programa usa os códigos ASCII,
# e o Alfabeto é tratado de forma circular, ou seja, 'Z' + 1 == 'A'.
# início do módulo principal
# descrição das variáveis utilizadas
# str palavra, codificada
# str char0, char1, char2, char3, char4, char5
# int deslocamento

# pré: palavra and palavra[i] em {'A'..'Z'} deslocamento

# Passo 1. Leia uma palavra com 6 caracteres maiúsculos e o deslocamento
# Passo 1.1. Leia uma palavra de 6 caracteres maiúsculos do alfabeto# Passo 1.2. Entre
# Passo 2. Codifique os caracteres da palavra
# Passo 2.1. Codifique o primeiro caractere
char0 =
# Passo 2.2. Codifique o segundo caractere
char1 =
# Passo 2.3. Codifique o terceiro caractere
char2 =
# Passo 2.4. Codifique o quarto caractere
char3 =
# Passo 2.6. Codifique o quinto caractere
char4 =
# Passo 2.7. Codifique o sexto caractere
char5 =
# Passo 2.8. Concatene os caracteres
codificada =
# Passo 3. Imprima a mensagem codificada

pós: codificada and codificada[i] em {'A'..'Z'} and
     para i em {0,...,5}:codificada[i]==palavra[i] + deslocamento
# fim do módulo principal
```

Como só temos ainda a estrutura sequencial para projetar algoritmos, só conseguiremos codificar eficientemente uma palavra com um número pequeno de caracteres. Anteriormente, descrevemos uma solução para codificar uma palavra de cinco (5) caracteres maiúsculos usando o cifrador de César. Adapte o programa para codificar uma palavra com 6 caracteres.

Exemplo 1

```
# formato da entrada
AAAAAA
2

# formato da saída
CCCCCC
```

Exemplo 2

```
# formato da entrada
ZZZZZZ
4

# formato da saída
DDDDDD
```