UNIRIO – Universidade Federal do Estado do Rio de Janeiro
Bacharelado em Sistemas de Informação
Técnicas de Programação 1 – 01/2020
Prof. Jobson Massollar

6ª Lista de Exercícios
Funções

Para cada exercício, crie um programa em C/C++ e implemente a função **main** para testar a sua função:

1. Implemente a função

   ```
   void segundo_grau(float a, float b, float c)
   ```

   que recebe os valores **a**, **b** e **c** de uma equação do segundo grau ($ax^2 + bx + c = 0$) e imprime as duas raízes da equação. Se não for possível calcular as duas raízes, então a função deve imprimir "Não há raízes".

2. Implemente a função

   ```
   int digitos(int n)
   ```

   que recebe um valor inteiro **n** e retorna o número de dígitos de **n**. Por exemplo, se n = 4875, a função deve retornar 4.

3. Implemente a função

   ```
   int conta_digitos(int n, int d)
   ```

   que recebe um valor inteiro **n** e retorna quantas vezes o dígito **d** (0 a 9) aparece no número **n**. Por exemplo, se n = 6764963 e d = 6 a função deve retornar 3.

4. Implemente a função

   ```
   int reverso(int n)
   ```

   que recebe um valor inteiro **n** e retorna o mesmo número com seus dígitos invertidos. Por exemplo, se n = 7631, a função deve retornar 1367.

5. Implemente a função

   ```
   int soma_divisores(int n)
   ```

   que retorna a soma dos divisores próprios de **n**. Dica: divisores próprios são os divisores de um número sem contar com ele mesmo. Os divisores próprios de 6 são 1, 2 e 3.

6. Implemente a função

   ```
   bool amigos(int x, int y)
   ```

   que retorna **true** ou **false** indicando se os números **x** e **y** são amigos ou não. **x** e **y** serão amigos se a soma dos divisores próprios de **x** for igual a **y** e se a soma dos divisores próprios de **y** for igual a **x**.

Dica 1: divisores próprios são os divisores de um número sem contar com ele mesmo. Os divisores próprios de 6 são 1, 2 e 3.

Dica 2: use a função implementada no exercício anterior.

Dica 3: 1184 e 1210 são amigos.

7. Implemente a função

```
void quebra_real(double n,
                 int &parte_inteira,
                 double &parte_decimal)
```

que recebe um valor real **n** e retorna sua parte inteira e sua parte decimal.

8. Implemente a função

```
bool colide_esfera(float x1, float y1, float r1,
                   float x2, float y2, float r2)
```

que retorna **true** se a esfera 1 de centro **(x1, y1)** e raio **r1** está em posição de colisão com a esfera 2 de centro **(x2, y2)** e raio **r2**, ou **false**, caso contrário.

9. Implemente a função

```
double cosseno(double x, int n)
```

que calcula o **cos(x)** com **n** parcelas usando a série a seguir (**x** está em radianos):

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \cdots$$

10. Implemente a função

```
void le_vetor_real(float v[], int tam)
```

para ler um vetor de números reais **v** de tamanho **tam**.

11. Implemente a função

```
int posicao(float v[], int tam, float n)
```

que recebe um vetor **v**, seu tamanho **tam** e um número **n** e retorna a posição de **n** em **v**. Se $\mathbf{n} \notin \mathbf{v}$ a função deve retornar **-1**. Dica: para ler o vetor use a função `le_vetor_real` implementada anteriormente.

12. Implemente as funções

```
float maior(float v[], int tam)
float menor(float v[], int tam)
```

que retorna o maior e o menor número em um vetor de números reais. Dica: para ler o vetor use a função `le_vetor_real` implementada anteriormente.

13. Implemente a função

```
bool pertence(int v[], int tam, float n)
```

que recebe um vetor **v**, seu tamanho **tam** e um número **n** e retorna **true** se **n** $\in$ **v**, ou **false**, caso contrário. Dica: para ler o vetor use a função `le_vetor` implementada em sala de aula.

14. Implemente a função

```
bool ordenado(int v[], int tam)
```

que retorna **true** se o vetor está ordenado ou **falso**, caso contrário. Dica: para ler o vetor use a função `le_vetor` implementada em sala de aula.

15. Implemente a função

```
int uniao(v1[], int tam1, int v2[], int tam2, int v[])
```

que recebe um vetor **v1** de tamanho **tam1** e um vetor **v2** de tamanho **tam2** e armazena no vetor **v** os elementos de **v1** $\cup$ **v2**. A função deve retornar a quantidade de números em **v**. Importante: o vetor **v** não pode ter valores repetidos, mesmo que **v1** ou **v2** tenham. Dica 1: use a função `pertence` implementada anteriormente. Dica 2: para ler o vetor use a função `le_vetor` implementada em sala de aula.

16. Implemente a função

```
int intersecao(v1[], int tam1, int v2[], int tam2, int v[])
```

que recebe um vetor **v1** de tamanho **tam1** e um vetor **v2** de tamanho **tam2** e armazena no vetor **v** os elementos de **v1** $\cap$ **v2**. A função deve retornar a quantidade de números em **v**. Importante: o vetor **v** não pode ter valores repetidos, mesmo que **v1** ou **v2** tenham. Dica 1: use a função `pertence` implementada anteriormente. Dica 2: para ler o vetor use a função `le_vetor` implementada em sala de aula.

17. Implemente a função

```
int conta_ocorrencias(int v[], int tam, int n)
```

que retorna o número de ocorrências de **n** em **v**. Dica: para ler o vetor use a função `le_vetor` implementada em sala de aula.

18. Implemente a função

```
bool sem_repeticao(int v[], int tam)
```

que recebe um vetor de inteiros **v** com **tam** elementos e retorna **true** se não há repetições em **v**, ou **false**, caso contrário. Dica: para ler o vetor use a função `le_vetor` implementada em sala de aula.

19. Implemente a função

```
int remove_repeticao(int v[], int tam)
```

que recebe um vetor de inteiros **v** com **tam** elementos e remove todos os valores repetidos de **v**. A função deve retornar a nova quantidade de inteiros em **v**. Dica: para ler o vetor use a função `le_vetor` implementada em sala de aula.

20. Implemente a função

```
int conta_letras(char str[])
```

que retorna a quantidade de letras (a..z ou A..Z) da string **str**.

21. Implemente a função

```
void retira_caracter(char str[], char c, char resultado[])
```

que remove todas as ocorrências do caracter **c** da string **str** e armazena a nova string em **resultado**.

22. Implemente a função

```
int retira_caracter(char str[], char c)
```

que remove todas as ocorrências do caracter **c** da string **str**.

23. Implemente a função

```
int substring(char str[], int ini, int fim, char resultado[])
```

para extrair a substring da posição **ini** até **fim** da string **str** e retornar na string **resultado**. A função deverá retornar o tamanho da string resultado ou -1 caso não seja possível extrair a substring.

24. Implemente a função

```
void plural(char palavra[])
```

que pluraliza a palavra de acordo com as seguintes regras:

- Se a palavra terminar em ‘l’ (ele) deve ser retirado o ‘l’ e acrescentado os caracteres ‘i’ e ‘s’ no final. Exemplo**:** "animal" → "animais".
- Se a palavra terminar em ‘r’ ou ‘s’ ou ‘z’ deve ser acrescentado os caracteres ‘e’ e ‘s’. Exemplos: "tambor" → "tambores", "feliz" → "felizes" e "viés" → "vieses".
- Se a palavra terminar em ‘m’ deve ser retirado o ‘m’ e acrescentado os caracteres ‘n’ e ‘s’. Exemplo: "homem" → "homens".
- Os demais casos deverão apenas receber o caractere ‘s’ no final. Exemplo: "casa" → "casas".

25. Pesquisa sobre as funções **srand** e **rand** da biblioteca do C/C++. Em seguida, implemente a função

```
int aleatorio(int a, int b)
```

que retorna um número aleatório entre **a** e **b**, inclusive.

26. Implemente a função

```
int segundos(int hora, int min, int seg)
```

que retorna a hora representada por **hora**, **min** e **seg** convertida em segundos.

27. Implemente a função

```
bool hora_valida(int hora, int min, int seg)
```

que verifica se os valores de **hora**, **min** e **seg** formam ou não uma hora válida no formato brasileiro (hora: 0 a 23, min: 0 a 59 e seg: 0 a 59).

28. Implemente a função

```
void incrementa_hora(int &hora, int &min, int &seg)
```

que incrementa a **hora**, **min** e **seg** (no formato brasileiro) em um segundo. Note que a hora 23:59:59 incrementada em um segundo vira 00:00:00.

29. Implemente a função

```
void decrementa_hora(int &hora, int &min, int &seg)
```

que decrementa a **hora**, **min** e **seg** (no formato brasileiro) de um segundo. Note que a hora 00:00:00 decrementada de um segundo vira 23:59:59.

30. Implemente a função

```
void incrementa_hora(int &hora, int &min, int &seg, int segundos)
```

que incrementa a **hora**, **min** e **seg** (no formato brasileiro) de acordo com os **segundos** fornecidos. Note que a hora 23:58:50 incrementada em 175 segundos vira 00:01:45.

31. Implemente a função

```
void decrementa_hora(int &hora, int &min, int &seg, int segundos)
```

que decrementa a **hora**, **min** e **seg** (no formato brasileiro) de acordo com os **segundos** fornecidos. Note que a hora 00:02:15 decrementada em 200 segundos vira 23:58:55.

32. Implemente a função

```
void duracao(int hora_ini, int min_ini, int seg_ini,
             int hora_fim, int min_fim, int seg_fim,
             int &horas, int &minutos, int &segundos)
```

que retorna a duração da hora inicial até a hora final em **horas**, **minutos** e **segundos**. Se a hora final for menor que a hora inicial é porque a hora inicial se refere a um dia e a hora final ao dia seguinte.

33. Implemente a função

```
double area(double vx[], double vy[], int n)
```

que recebe **n** vértices de um polígono e calcula a área desse polígono usando a fórmula:

$$area = \frac{\sum_{i=1}^{n} x_i \times y_{i+1} - x_i \times y_{i-1}}{2}$$

onde:

i+1 = n se i = n

i-1 = 1 se i = 0

Os vetores **vx** e **vy** armazenam as coordenadas dos vértices da seguinte forma:

v_1 = (vx[0], vy[0])

v_2 = (vx[1], vy[1])

v_3 = (vx[2], vy[2])

...

v_n = (vx[n-1], vy[n-1])

34. Implemente a função

```
bool encaixa(int a, int b)
```

que retorna **true** se o número **b** corresponde aos <u>últimos</u> dígitos de **a**; ou **false**, caso contrário. Exemplos:

a = 567890 b = 890 → true

a = 1243 b = 1243 → true

a = 2357 b = 358 → false

a = 2345 b = 12345 → false

Desafios

35. Implemente a função

```
bool eh_segmento(int a, int b)
```

que retorna **true** se o menor número entre **a** e **b** faz parte do outro número; ou **false**, caso contrário. <u>Dica</u>: use a função `encaixa` implementada em um exercício anterior. Exemplos:

a = 567890 b = 678 → true

a = 17 b = 1754 → true

a = 2357 b = 358 → false

a = 12345 b = 45 → true

36. Faça um programa que implemente um jogo de Craps. O jogador lança um par de dados, obtendo um valor de 2 a 12. Se na primeira jogada ele tira 7 ou 11 o jogo termina e ele ganha. Porém, se na primeira jogada ele tira 2, 3 ou 12, o jogo termina e ele perde. Para qualquer outro valor (4, 5, 6, 8 9 ou 10) esta é a pontuação do jogador. O objetivo agora é continuar jogando até que o jogador tire novamente a mesma pontuação. Entretanto, o jogador perde se tirar um 7 antes de tirar a mesma pontuação. Implemente, pelo menos, duas funções:

`int lancar_dado()`: que simula o lançamento de um único dado.

`int jogar_dados()`: que simula o lançamento de dois dados simultaneamente.

37. Quando jogamos um único dado, qualquer um dos 6 valores tem a mesma probabilidade de sair. Essa mesma lógica vale para quando jogamos dois dados, ou seja, os valores de 2 a 12 tem a mesma probabilidade de sair? Para responder a essa pergunta, simule o arremesso de dois dados 1 milhão de vezes e apresente uma estatística do percentual de vezes que cada valor saiu.

38. Implemente a função

```
bool cpf_valido(long long cpf)
```

que recebe um número de CPF e retorna **true** se o CPF é válido; ou **false**, caso contrário. Um CPF é válido se obedece às seguintes regras:

- Possui exatamente 11 dígitos
- Os 11 dígitos não podem ser todos iguais (tudo 1 ou tudo 2, por exemplo)
- OS 9 primeiros dígitos compõem o número do CPF e os 2 últimos dígitos são os dígitos verificadores (DV). Os DV são calculados da seguinte forma:

  J é o **1° dígito verificador** do CPF.

  K é o **2° dígito verificador** do CPF.

  Primeiro Dígito

  Para obter **J** multiplicamos os 9 primeiros dígitos do CPF (A a I) pelas constantes da tabela a seguir:

A	B	C	D	E	F	G	H	I
x 10	x 9	x 8	x 7	x 6	x 5	x 4	x 3	x 2

  O resultado da soma 10A + 9B + 8C + 7D + 6E + 5F + 4G + 3H + 2I é **dividido por 11.** Analisamos então o **resto** dessa divisão: se for 0 ou 1, o dígito **J** é **0** (zero). Se for 2, 3, 4, 5, 6, 7, 8, 9 ou 10, o dígito **J** é **11 – RESTO**.

  Segundo Dígito

  Já temos **J**. Para obter **K** multiplicamos os 10 primeiros dígitos do CPF (A a J) pelas constantes da tabela a seguir:

A	B	C	D	E	F	G	H	I	J
x 11	x 10	x 9	x 8	x 7	x 6	x 5	x 4	x 3	x 2

  O resultado da soma 11A + 10B + 9C + 8D + 7E + 6F + 5G + 4H + 3I + 2J é **dividido por 11.** Analisamos então o **resto** dessa divisão: se for 0 ou 1, o dígito **K** é **0** (zero). Se for 2, 3, 4, 5, 6, 7, 8, 9 ou 10, o dígito **K** é **11 – RESTO**.