Departamento de Informática – PUC-Rio
INF1620 - Estruturas de Dados

Primeira Lista de Exercícios – 2005.1

1. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que verifique se um determinado número é primo. O número a ser testado deve ser fornecido pelo usuário do programa. O programa deve imprimir a mensagem "NUMERO PRIMO" caso o número seja primo, e "NUMERO NAO PRIMO" caso contrário.

2. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que imprima os n primeiros termos da série de Fibonacci, lembrando que esta série é definida da seguinte forma:

$$\begin{aligned} termo_1 &= 1 \\ termo_2 &= 1 \\ termo_n &= termo_{n-1} + termo_{n-2} \end{aligned}$$

   O usuário do programa deve fornecer o número de termos, e estes devem ser impressos separados por um espaço. Por exemplo, se o usuário pedir os 6 primeiros termos da série de Fibonacci, o programa deve ter a seguinte saída:

   1 1 2 3 5 8

3. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que calcule e imprima a soma dos n primeiros números naturais ímpares. O usuário do programa deve fornecer quantos números devem ser somados.

4. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que calcule e imprima a soma dos n primeiros números naturais pares. O usuário do programa deve fornecer quantos números devem ser somados.

5. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que, dado um número inteiro n, imprima seu maior divisor x , onde $x < n$. O número n a ser testado deve ser fornecido pelo usuário do programa.

6. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que imprima a soma dos n primeiros termos da série de Fibonacci, lembrando que esta série é definida da seguinte forma:

$$\begin{aligned} termo_1 &= 1 \\ termo_2 &= 1 \\ termo_n &= termo_{n-1} + termo_{n-2} \end{aligned}$$

7. Faça um programa completo (função `main` e inclusão dos arquivos de cabeçalhos adequados) que teste se um número pertence a série de Fibonacci, lembrando que esta série é definida da seguinte forma:

$$\begin{aligned} termo_1 &= 1 \\ termo_2 &= 1 \\ termo_n &= termo_{n-1} + termo_{n-2} \end{aligned}$$

O número a ser testado deve ser fornecido pelo usuário do programa. O programa deve imprimir a mensagem "PERTENCE A SERIE" caso o número pertença a série, e "NAO PERTENCE A SERIE" caso contrário.

8. Implemente a função `raizes`, que calcula as raízes de uma equação do segundo grau, do tipo $ax^2 + bx + c = 0$. Essa função deve obedecer o protótipo

```
void raizes (float a, float b, float c, float * x1, float * x2);
```

onde `a`, `b` e `c` representam os coeficientes da equação, e `x1` e `x2` são ponteiros para as variáveis onde devem ser guardadas as raízes da equação.
Observações:

(a) `x1` deve guardar a raiz de menor valor e `x2` a de maior valor.

(b) Assuma que a equação sempre tem raízes distintas.

(c) Para o cálculo da raiz quadrada de um número, utilize a função `sqrt` definida na biblioteca padrão de funções do C. Essa função está definida no arquivo de cabeçalhos `math.h` e tem o protótipo

```
double sqrt (double n);
```

9. Implemente a função `calc_circulo`, que calcula a área e a circunferência de um círculo de raio r. Essa função deve obedecer o protótipo

```
void calc_circulo(float r, float * circunferencia, float * area);
```

Fórmulas:

$$\begin{aligned} A &= \pi r^2 \\ c &= 2\pi r \\ \pi &= 3.14159265 \end{aligned}$$

10. Implemente a função `calc_esfera`, que calcula o volume e a área da superfície de uma esfera de raio r. Essa função deve obedecer o protótipo

```
void calc_esfera(float r, float * area, float * volume);
```

Fórmulas:

$$\begin{aligned} V &= \frac{4}{3}\pi r^3 \\ A &= 4\pi r^2 \\ \pi &= 3.14159265 \end{aligned}$$

11. Implemente a função `calc_cilindro`, que calcula o volume e a área da superfície de um cilindro de raio r e altura h. Essa função deve obedecer o protótipo

```
void calc_cilindro(float r, float h, float * area, float * volume);
```

Fórmulas:

$$\begin{aligned} V &= \pi r^2 h \\ A &= 2\pi r h \ + \ 2\pi r^2 \\ \pi &= 3.14159265 \end{aligned}$$

12. Implemente a função `calc_cone`, que calcula o volume e a área da superfície de um cone de raio r e altura h. Essa função deve obedecer o protótipo

```
void calc_cone(float r, float h, float * area, float * volume);
```

Fórmulas:

$$\begin{aligned} V &= \frac{1}{3}\pi r^2 h \\ A &= \pi r\sqrt{r^2 \ + \ h^2} \\ \pi &= 3.14159265 \end{aligned}$$

Obs: Para o cálculo da raiz quadrada de um número, utilize a função `sqrt` definida na biblioteca padrão de funções do C. Essa função está definida no arquivo de cabeçalhos `math.h` e tem o protótipo

```
double sqrt (double n);
```

13. Implemente a função `raizes`, que calcula as raízes de uma equação do segundo grau, do tipo $ax^2 \ + \ bx \ + \ c \ = \ 0$. Essa função deve obedecer o protótipo

```
int raizes (float a, float b, float c, float * x1, float * x2);
```

onde `a`, `b` e `c` representam os coeficientes da equação, e `x1` e `x2` são ponteiros para as variáveis onde devem ser guardadas as raízes da equação. A função deve retornar o número de raízes reais que a equação possui.
Observações:

(a) Se as raízes forem reais e distintas,`x1` deve guardar a raiz de menor valor e `x2` a de maior valor e a função deve retornar `2`.

(b) Se as raízes forem reais e iguais $x1 \ = \ x2$ e a função deve retornar `1`.

(c) Se não existirem raízes reais, $x1 \ = 0$ e $x2 \ = 0$ e a função deve retornar `0`.

(d) Para o cálculo da raiz quadrada de um número, utilize a função `sqrt` definida na biblioteca padrão de funções do C. Essa função está definida no arquivo de cabeçalhos `math.h` e tem o protótipo

```
double sqrt (double n);
```

14. Implemente a função `calc_paralelepipedo`, que calcula a área e o volume de um paralelepípedo de lados a, b, c. Essa função deve obedecer o protótipo

```
void calc_paralelepipedo(float a, float b, float c, float * area, float * volume);
```

Fórmulas:

$$\begin{aligned} A &= 2(ab + ac + bc) \\ V &= abc \end{aligned}$$

15. Implemente a função `calc_calota`, que calcula o volume e a área da superfície de uma calota esférica de raio r e altura h. Essa função deve obedecer o protótipo

```
void calc_calota(float r, float h, float * area, float * volume);
```

Fórmulas:

$$\begin{aligned} V &= \frac{1}{3}\pi h^2(3r - h) \\ A &= 2\pi r h \\ \pi &= 3.14159265 \end{aligned}$$

16. Implemente a função `calc_tronco`, que calcula o volume e a área da superfície de um tronco de cone de raios a, b e altura h. Essa função deve obedecer o protótipo

```
void calc_tronco(float a, float b, float h, float * area, float * volume);
```

Fórmulas:

$$\begin{aligned} A &= \pi(a + b)\sqrt{h^2 + (b - a)^2} \\ V &= \frac{1}{3}\pi h\,(a^2 + ab + b^2) \\ \pi &= 3.14159265 \end{aligned}$$

Obs: Para o cálculo da raiz quadrada de um número, utilize a função `sqrt` definida na biblioteca padrão de funções do C. Essa função está definida no arquivo de cabeçalhos `math.h` e tem o protótipo

```
double sqrt (double n);
```

17. Implemente a função `negativos`, que recebe como parâmetro um vetor de números de ponto flutuante (`vet`) de tamanho `n` e retorna quantos números negativos estão armazenados nesse vetor. Essa função deve obedecer o protótipo:

```
int negativos (int n, float * vet);
```

18. Implemente a função `positivos`, que recebe como parâmetro um vetor de números de ponto flutuante (`vet`) de tamanho `n` e retorna quantos números positivos estão armazenados nesse vetor. Essa função deve obedecer o protótipo:

```
int positivos (int n, float * vet);
```

19. Implemente a função `pares`, que recebe como parâmetro um vetor de números inteiros (`vet`) de tamanho `n` e retorna quantos números pares estão armazenados nesse vetor. Essa função deve obedecer o protótipo:

```
int pares (int n, int * vet);
```

20. Implemente a função `impares`, que recebe como parâmetro um vetor de números inteiros (`vet`) de tamanho `n` e retorna quantos números ímpares estão armazenados nesse vetor. Essa função deve obedecer o protótipo:

```
int impares (int n, int * vet);
```

21. Implemente a função `avalia`, que permite a avaliação de polinômios. Cada polinômio é definido por um vetor contendo seus coeficientes. Por exemplo, o polinômio de grau 2, $3x^2 + 2x + 12$, terá um vetor de coeficientes igual a $v[\,] = \{12, 2, 3\}$. A função `avalia` deve obedecer o protótipo:

```
double avalia(double * poli, int grau, double x);
```

    Onde o parâmetro `poli` é o vetor com os coeficientes do polinômio, `grau` é o grau do polinômio, e `x` é o valor para o qual o polinômio deve ser avaliado.

22. Implemente a função `deriva`, que calcula a derivada de um polinômio. Cada polinômio é definido por um vetor contendo seus coeficientes. Por exemplo, o polinômio de grau 2, $3x^2 + 2x + 12$, terá um vetor de coeficientes igual a $v[\,] = \{12, 2, 3\}$. A função `deriva` deve obedecer o protótipo:

```
void deriva(double * poli, int grau, double * out);
```

    Onde o parâmetro `poli` é o vetor com os coeficientes do polinômio a ser derivado, `grau` é o grau desse polinômio, e `out` é o vetor no qual a função deve guardar os coeficientes do polinômio resultante da derivada.

23. Implemente a função `max_vet`, que recebe como parâmetro um vetor de números de ponto flutuante (`vet`) de tamanho `n` e retorna o maior número armazenado nesse vetor. Essa função deve obedecer o protótipo:

```
float max_vet (int n, float * vet);
```

24. Implemente a função `min_vet`, que recebe como parâmetro um vetor de números de ponto flutuante (`vet`) de tamanho `n` e retorna o menor número armazenado nesse vetor. Essa função deve obedecer o protótipo:

```
float min_vet (int n, float * vet);
```

25. Implemente a função `busca`, que recebe como parâmetro um vetor de números inteiros (`vet`) de tamanho `n` e um valor `x`. A função deve retornar `1` se `x` pertence a esse vetor e `0` caso contrário. Essa função deve obedecer o protótipo:

```
int busca (int n, int * vet, int x);
```

26. Implemente a função `maiores`, que recebe como parâmetro um vetor de números inteiros (`vet`) de tamanho `n` e um valor `x`. A função deve retornar quantos números maiores do que `x` existem nesse vetor. Essa função deve obedecer o protótipo:

```
int maiores(int n, int * vet, int x);
```

27. Implemente a função `menores`, que recebe como parâmetro um vetor de números inteiros (`vet`) de tamanho `n` e um valor `x`. A função deve retornar quantos números menores do que `x` existem nesse vetor. Essa função deve obedecer o protótipo:

```
int menores(int n, int * vet, int x);
```