# Algoritmos e Programação I: Lista de Exercícios 06 - Orientações para submissão no Moodle e BOCA. *

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

Após submeter no BOCA, não se esqueça de enviar o programa para o Moodle, usando o seguinte nome: pxxloginname.c, onde xx é o número do problema e o loginname é o seu login.

Compile o seu programa usando:

```
gcc -Wall -pedantic -std=c99 -o programa programa.c [-lm]
```

A flag `-lm` deve ser usada quando o programa incluir a biblioteca `math.h`.

Uma forma eficiente de testar se o seu programa está correto é gerando arquivos de entrada e saída e verificar a diferença entre eles. Isso pode ser feito da seguinte forma:

```
./programa < programa.in > programa.out
```

O conjunto de entrada deve ser digitado e salvo num arquivo `programa.in`. O modelo da saída deve ser digitado e salvo num arquivo `programa.sol`.

Verifique se a saída do seu programa `programa.out` é EXATAMENTE igual ao modelo de saída `programa.sol`. Isso pode ser feito com a utilização do comando `diff`, que verifica a diferença entre dois arquivos.

```
diff programa.out programa.sol
```

O seu programa só está correto se o resultado de `diff` for vazio (e se você fez todos os passos como indicado).

A solução dos exercícios deve seguir a metodologia descrita em sala:

a. Diálogo

b. Saída/Entrada (pós e pré)

c. Subdivisão

d. Abstrações

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina (B. Forouzan e R. Gilbert, A. B. Tucker et al. e S. Leestma e L. Nyhoff, Lambert et al., H. Farrer et al., K. B. Bruce et al., e Material Didático dos Profs. Edson Takashi Matsubara e Fábio Viduani Martinez.

e. Implementação

f. Teste

Na submissão ao BOCA não se esqueça de comentar todos os `printf`'s da entrada e que a saída não pode conter acentuação ou *ç*.

# Exercícios

1. (somabits.c)[fhvm]

   a. Escreva uma função com a seguinte interface:

   ```
   int somabit(int b1, int b2, int* vaium)
   ```

   que receba três dígitos binários e retorne um dígito binário representando a soma dos três bits, e devolva também um outro dígito binário, no parâmetro `vaium`, representando o valor do vai-um desta operação.

   b. Escreva um programa que recebe um inteiro $n > 0$ e uma sequência de $n$ pares de números inteiros na base binária e, usando a função do item (a), calcule, para cada par, um número na base binária que é a soma dos dois números fornecidos.

   Exemplo de entrada:

   ```
   somabits.in
   3
   100 1
   111111 1
   10101 1010
   ```

   Exemplo de saída:

   ```
   somabits.sol
   101
   1000000
   11111
   ```

2. (subsequencia.c)[fhvm] **Subsequência**

   a. Escreva uma função com a seguinte interface:

   ```
   int sufixo(int a, int b)
   ```

   que receba dois números inteiros positivos $a$ e $b$ e verifique se $b$ é um sufixo de $a$. Em caso positivo, a função deve retorna 1; caso contrário, a função deve devolver 0.

   Exemplo:

   | $a$ | $b$ | | |
   |---|---|---|---|
   | 567890 | 890 | $\rightarrow$ | sufixo |
   | 1234 | 1234 | $\rightarrow$ | sufixo |
   | 2457 | 245 | $\rightarrow$ | não é sufixo |
   | 457 | 2457 | $\rightarrow$ | não é sufixo |

b. Usando a função do item anterior, escreva um programa que receba um número inteiro $k > 0$ e uma lista de $k$ pares de números inteiros positivos $a$ e $b$ e verifique, para cada par, se o menor deles é subsequência do outro. Em caso positivo, imprima $a$ `S` $b$; em caso negativo, imprima `N`.

Exemplo:

Para $n = 3$ e os seguintes pares de números, temos

| $a$ | $b$ | | |
|---|---|---|---|
| 567890 | 678 | $\rightarrow$ | $b$ é subsequência de $a$ |
| 1234 | 2212345 | $\rightarrow$ | $a$ é subsequência de $b$ |
| 235 | 236 | $\rightarrow$ | um não é subsequência do outro |

Exemplo de entrada:

```
sequencia.in
3
567890 678
1234 2212345
235 236
```

Exemplo de saída:

```
sequencia.sol
678 S 567890
1234 S 2212345
N
```

3. (logaritmo.c)[fhvm] **Logaritmo na base 2** O **piso** de um número $x$ é o único inteiro $i$ tal que $iv \leq x < i + 1$. O piso de $x$ é denotado por $\lfloor x \rfloor$. Aplicando o piso na função $\log_2$, obtemos a seguinte conjunto de valores:

| $n$ | 15 | 16 | 31 | 32 | 63 | 64 | 127 | 128 | 255 | 256 | 511 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lfloor \log_2 n \rfloor$ | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 7 | 7 | 8 | 8 | 9 |

a. Escreva uma função com a seguinte interface:

```
int piso_log2(int n)
```

que receba um número inteiro $n \geq 1$ e devolva $\lfloor \log_2 n \rfloor$.

b. Escreva um programa que receba um número inteiro $k > 0$ e uma sequência de $k$ números inteiros positivos e, para cada número $n$ fornecido, calcule e imprima $\lfloor \log_2 n \rfloor$.

Exemplo de entrada:

```
logaritmo.in
3
16
31
32
```

Exemplo de saída:

```
logaritmo.sol
4
4
5
```

4. (malternante.c)[fhvm] Uma sequência de $n$ números inteiros não nulos é defina como $m$-**alternante** se é constituída por $m$ segmentos: o primeiro com um elemento, o segundo com dois elementos e assim por diante até o $m$-ésimo, com $m$ elementos. Além disso, os elementos de um mesmo segmento devem ser todos pares ou todos ímpares e para cada segmento, se seus elementos forem todos pares (ímpares), os elementos do segmento seguinte devem ser todos ímpares (pares). Por exemplo:

   - A sequência com $n = 10$ elementos: $\underline{8}\ \underline{3\ 7}\ \underline{2\ 10\ 4}\ \underline{5\ 13\ 9\ 11}$ é 4-alternante.
   - A sequência com $n = 3$ elementos: $\underline{7}\ \underline{2\ 8}$ é 2-alternante.
   - A sequência com $n = 8$ elementos: $\underline{1}\ \underline{12\ 4}\ \underline{3\ 13\ 5}\ \underline{8\ 6}$ não é alternante, pois o último segmento não tem tamanho 4.

   a. Escreva uma função com a seguinte interface:

   ```
   int bloco(int m)
   ```

   que receba um número inteiro $m > 0$ e leia $m$ números inteiros, retornando um dos seguintes valores:

   $0$, se os $m$ números lidos forem pares;
   $1$, se os $m$ números lidos forem ímpares;
   $-1$, se entre os $m$ números lidos há números com paridades diferentes.

   b. Usando a função do item anterior, escreva um programa que receba um número inteiro $k > 0$ que representa o número de casos de teste. Para cada caso de teste, receba um número inteiro $n > 0$ e mais $n$ números inteiros, e verifique se a sequência é $m$-alternante. Em caso positivo, imprima o valor de $m$. Caso contrário, imprima 0.

Exemplo de entrada:

```
malternante.in
3
10 8 3 7 2 10 4 5 13 9 11
3 7 2 8
8 1 12 4 3 13 5 8 6
```

Exemplo de saída:

```
malternante.sol
4
2
0
```

5. (inverte.c)[fhvm] **Inverta e adicione** Um número inteiro positivo é chamado de **palíndromo** se lido da esquerda para a direita e da direita para a esquerda representa o mesmo número.

   A função `inverta e adicione` é curiosa e divertida. Iniciamos com um número inteiro positivo, invertemos seus dígitos e adicionamos o número invertido ao original. Se o resultado da adição não for um palíndromo, repetimos esse processo até obtermos um número palíndromo.

   Por exemplo, se começarmos com o número 195, obteremos o número 9339 como o palíndromo resultante desse procedimento após 4 adições:

$$\begin{array}{rr} & 195 \\ + & 591 \\ \hline & 786 \end{array} \quad + \quad \begin{array}{rr} & 786 \\ + & 687 \\ \hline & 1473 \end{array} \quad + \quad \begin{array}{rr} & 1473 \\ + & 3741 \\ \hline & 5214 \end{array} \quad + \quad \begin{array}{rr} & 5214 \\ + & 4125 \\ \hline & 9339 \end{array}$$

   O procedimento acima levará a palíndromos em poucos passos para quase todos os números inteiros positivos. Mas existem exceções interessantes. O número 196 é o primeiro número para o qual nenhum palíndromo foi encontrado por este procedimento. Entretanto, nunca foi provado que tal palíndromo não existe.

   a. Escreva uma função com a seguinte interface:

   ```
   unsigned int inverte(unsigned int n)
   ```

   que receba um número inteiro positivo $n$ e devolva esse número invertido.

   b. Escreva um programa que receba um número inteiro $k > 0$ que representa a quantidade de casos de teste. Para cada caso de teste, receba um número inteiro $n > 0$ e compute um número inteiro $p$ através da função `inverta e adicione` descrita acima, mostrando na saída o número mínimo de adições para encontrar o palíndromo e também o próprio palíndromo. Caso um número inteiro de entrada use mais que 1.000 iterações/adições ou que o número gerado pelo processo ultrapasse o valor 4.294.967.295, escreva ? ? na saída.

   Exemplo de entrada:

   ```
   inverte.in
   4
   195
   265
   750
   196
   ```

   Exemplo de saída:

   ```
   inverte.sol
   4 9339
   5 45254
   3 6666
   ? ?
   ```

6. (mmc.c)[fhvm] O **mínimo múltiplo comum** entre dois números inteiros positivos pode ser calculado conforme o esquema abaixo ilustrado:

| | | |
|---|---|---|
| 8 | 36 | 2 |
| 4 | 18 | 2 |
| 2 | 9 | 2 |
| 1 | 9 | 3 |
| 1 | 3 | 3 |
| 1 | 1 | $2^3 \cdot 3^2 = 72$ |

   a. Escreva uma função com a seguinte interface:

```
int divisao(int* m, int* n, int d)
```

   que receba três números inteiros positivos $m, n$ e $d$ e retorne 1 se $d$ divide $m$, $n$ ou ambos, e 0, caso contrário. Além disso, em caso positivo, a função deve devolver, nos parâmetros correspondentes, um valor que representa o quociente da divisão de $m$ por $d$ e outro valor que representa o quociente da divisão de $n$ por $d$.

   b. Escreva um programa que receba um número inteiro $k > 0$ e uma sequência de $k$ pares de números inteiros positivos $m$ e $n$ e calcule, usando a função do item (a), o mínimo múltiplo comum entre $m$ e $n$.

   Exemplo de entrada:

```
mmc.in
3
8 36
12 54
75 42
```

   Exemplo de saída:

```
mmc.sol
72
108
1050
```

7. (palindromo.c)[fhvm] Dizemos que um número natural $n$ é **palíndromo** se lido da esquerda para direita e da direita para esquerda é o mesmo número.

   Exemplos:

   567765 é palíndromo.
   32423 é palíndromo.
   567675 não é palíndromo.

   a. Escreva uma função com a seguinte interface:

```
void quebra(int n, int* prim, int* ult, int* miolo)
```

que receba um número inteiro $n > 0$ e devolva três números inteiros nos parâmetros correspondentes: o primeiro dígito de $n$, o último dígito de $n$ e um inteiro que represente o número $n$ sem seu primeiro e último dígitos.

Exemplo:

| valor inicial de $n$ | primeiro dígito | último dígito | miolo de $n$ |
|---|---|---|---|
| 732 | 7 | 2 | 3 |
| 14738 | 1 | 8 | 473 |
| 78 | 7 | 8 | 0 |
| 7 | 7 | 7 | 0 |

b. Usando a função do item (a), escreva um programa que receba um número inteiro $k > 0$ e uma sequência de $k$ números inteiros positivos e verifique, para cada um deles, se é palíndromo, imprimindo `P` em caso positivo e `N` em caso negativo. Suponha que os números da sequência não contém o dígito 0.

Exemplo de entrada:

```
palindromo.in
3
567765
32423
567675
```

Exemplo de saída:

```
palindromo.sol
P
P
N
```

Nos exercícios a seguir são usadas aproximações e geração de números aleatórios:

8. (raiz.c)[fhvm] **Raiz quadrada** Os babilônios descreveram há mais de 4 mil anos um método para calcular a raiz quadrada de um número. Esse método ficou posteriormente conhecido como método de Newton. Dado um número $x$, o método parte de um chute inicial $y$ para o valor da raiz quadrada de $x$ e sucessivamente encontra aproximações desse valor, calculando a média aritmética de $y$ e de $x/y$. O exemplo a seguir mostra o método em funcionamento para o cálculo da raiz quadrada de 3, com chute inicial 1:

| $x$ | $y$ | $x/y$ | $(y + x/y)/2$ |
|---|---|---|---|
| 3 | 1 | 3 | 2 |
| 3 | 2 | 1.5 | 1.75 |
| 3 | 1.75 | 1.714286 | 1.732143 |
| 3 | 1.732143 | 1.731959 | 1.732051 |
| 3 | 1.732051 | 1.732051 | 1.732051 |

O valor retornado é o valor de $y$, isto é, $y = \sqrt{x}$.

Podemos descrever esta fórmula de uma outra maneira:

$$\begin{cases} y_0 = \alpha\,, \\ y_i = \dfrac{y_{i-1} + x/y_{i-1}}{2}\,, \quad \text{para } i \geq 1\,, \end{cases}$$

onde $\alpha$ é um chute inicial e $y_{i^*} = \sqrt{x}$, para algum $i^* \geq 1$ e $|y_{i^*} - y_{i^*-1}| < \varepsilon$.

a. Escreva uma função com a seguinte interface:

```
double raiz(double x, double epsilon, int* passos)
```

que receba um número real $x$ e um número real $\varepsilon$, com $0 < \varepsilon < 1$, e devolva o valor de $\sqrt{x}$ usando o método de Newton descrito acima, até que o valor absoluto da diferença entre dois valores consecutivos de $y$ seja menor que $\varepsilon$. A função deve retornar também, no parâmetro `passos`, a quantidade de passos realizados para obtenção da raiz de $x$ com precisão $\varepsilon$.

b. Escreva um programa que receba um número inteiro $k > 0$ que representa a quantidade de casos de teste. Para cada caso de teste, receba um número real positivo $x$ e um número real $\varepsilon$, com $0 < \varepsilon < 1$, e calcule e imprima $\sqrt{x}$ com precisão dada por $\varepsilon$. Mostre $\sqrt{x}$ com 6 casas decimais na saída.

Exemplo de entrada:

```
raiz.in
2
3 0.000001
4 0.1
```

Exemplo de saída:

```
raiz.sol
1.732051 5
2.000610 3
```

9. (integral.c)[fhvm] **Integral do cosseno** Podemos calcular a o valor da integral da função cosseno no intervalo $(a, b)$ usando o método dos trapézios, descrito a seguir:

$$\int_a^b \cos(x) = \frac{h}{2}\Big(\cos(a) + 2\cos(a+h) + 2\cos(a+2h) + \ldots + 2\cos(a+(n-1)h) + \cos(b)\Big),$$

onde

- $a$ e $b$ definem o intervalo de integração, com $-\pi/2 \leq a < b \leq \pi/2$;
- $n$ é o número de sub-intervalos, com $n \geq 1$;
- $h$ é uma constante, determinada por $h = (b - a)/n$.

a. Escreva uma função com a seguinte interface:

```
double cosseno(double x, double epsilon)
```

que receba um número real $x$ que representa um arco em radianos e um número real $\varepsilon$ que representa a precisão desejada, onde $0 < \varepsilon < 1$, e use a seguinte série para calcular o cosseno de $x$:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} - \ldots + (-1)^k \frac{x^{2k}}{(2k)!} + \ldots$$

A função deve retornar a aproximação do valor do cosseno de $x$ até que o termo $x^{2k}/(2k)!$ seja menor que $\varepsilon$.

b. Escreva um programa que que receba um número inteiro $k > 0$ que indica o número de casos de teste. Para cada caso de teste, receba dois números reais $a$ e $b$ que definem o intervalo de integração $(a, b)$, onde $-\pi/2 \leq a < b \leq \pi/2$, um número inteiro $n$ que representa o número de sub-intervalos e um número real $\varepsilon$ que determina a precisão do cálculo da função cosseno, onde $0 < \varepsilon < 1$, integre numericamente a função cosseno sobre o intervalo $(a, b)$ usando $n$ trapézios, e imprima o valor da integral da função cosseno no intervalo $(a, b)$ com precisão de 8 casas decimais. Use a função do item (a).

Exemplo de entrada:

```
integral.in
2
0 1.57 5 .00000001
-0.78 0 50 .00000001
```

Exemplo de saída:

```
integral.sol
0.99176982
0.70326516
```

10. (`areatrap.c`) No exemplo para calcular a área sob uma curva, consideramos aproximação numérica de integrais usando retângulos. Como na Figura 1 indica, uma aproximação melhor pode ser obtida usando trapézios ao invés de retângulos.

    A soma das áreas desses trapézios é dada por

    A soma das áreas dos trapézios da Figura 1 é dada por

$$\sum_{i=1}^{n} [f(x_{i-1}) + f(x_i)] \frac{\Delta x}{2}$$

    onde $\Delta x = (b - a)/n$ .

    A expressão acima também pode ser escrita como

$$\frac{\Delta x}{2} [f(x_0) + 2f(x_1) + \cdots 2f(x_{n-1}) + f(x_n)]$$

ou

$$\Delta x \left[ \frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i) \right]$$

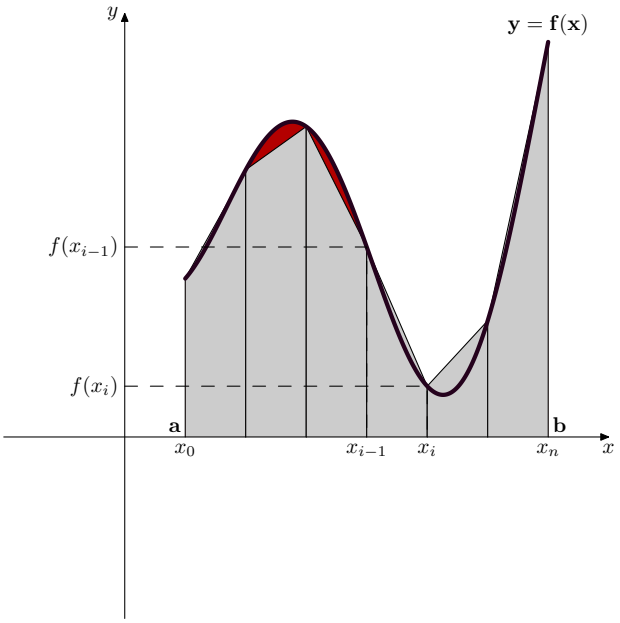


Figura 1: Método dos Trapézios

Escreva um programa para aproximar uma integral usando este método dos trapézios.

O programa deve ser implementado usando as funções com as seguintes interfaces:

```
void ledados(float* a, float* b, unsigned int* n)
float calculaareatrap(float a, float b, unsigned int n);
float f(float x);
void imprimeresultado(unsigned int n, float y);
```

A função `ledados` lê os valores do intervalo $[a, b]$ e o número de subintervalos utilizados na aproximação. A função `calculaareatrap` recebe os valores de $a$, $b$ e $n$ e utilizando o método dos trapézios calcula o valor aproximado da área sob a curva da função (especificada em `f`) e o eixo $x$. no intervalo $[a, b]$ utilizando $n$ subintervalos. A função `imprimeresultado` imprime o valor da área.

11. `(areasimpson.c)` Outro método de integração numérica que em geral dá uma aproximação melhor que o método dos retângulos ou o método dos trapézios descritos anteriormente é baseado no uso de parábolas e é conhecido com Regra de Simpson. Neste método, o intervalo $[a, b]$ é dividido em um número par de $n$ subintervalos, cada um de comprimento $\Delta x$, e a soma

$$\frac{\Delta x}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

é usada para aproximar a integral de $f$ no intervalo $[a, b]$. Escreva um programa para aproximar uma integral usando a Regra de Simpson.

O programa deve ser implementado usando as funções com as seguintes interfaces:

```
void ledados(float* a, float* b, unsigned int* n)
float calculaareasimpson(float a, float b, unsigned int n);
float f(float x);
void imprimeresultado(unsigned int n, float y);
```

A função `ledados` lê os valores do intervalo $[a, b]$ e o número de subintervalos utilizados na aproximação. A função `calculaareasimpson` recebe os valores de $a$, $b$ e $n$ e utilizando o método de Simpson calcula o valor aproximado da área sob a curva da função (especificada em `f`) e o eixo $x$. no intervalo $[a, b]$ utilizando $n$ subintervalos. A função `imprimeresultado` imprime o valor da área.

12. `(zerobisec.c)` Muitas vezes é necessário encontrar um zero para uma função $f$, isto é, um valor $c$ onde $f(c) = 0$. Geometricamente, estamos procurando por um ponto $c$ no eixo $x$ na qual o gráfico de $y = f(x)$ corta o eixo $x$. Se $f$ é uma função contínua entre $x = a$ e $x = b$, isto é não existe interrupção no gráfico $y = f(x)$ entre estes dois valores, e $f(a)$ e $f(b)$ possuem sinais opostos, então $f$ deve ter pelo menos um zero entre $x = a$ e $x = b$. Um método para encontrar um tal zero, ou pelo menos uma aproximação dele, é o método da bisseção. Para isso, divida ao meio o intervalo $[a, b]$ e determine em qual metade $f$ troca de sinal; então $f$ deve ter um zero nessa metade de intervalo. Agora divida ao meio o subintervalo e determine em qual metade $f$ troca de sinal. Repetindo este processo dá uma sequência de subintervalos cada vez menores, cada um deles contém um zero da função. Veja a Figura 2. O processo pode ser finalizado quando um pequeno subintervalo, digamos de comprimento menor que 0.0001, é obtido, ou $f$ tem o valor 0 em um dos extremos do subintervalo.

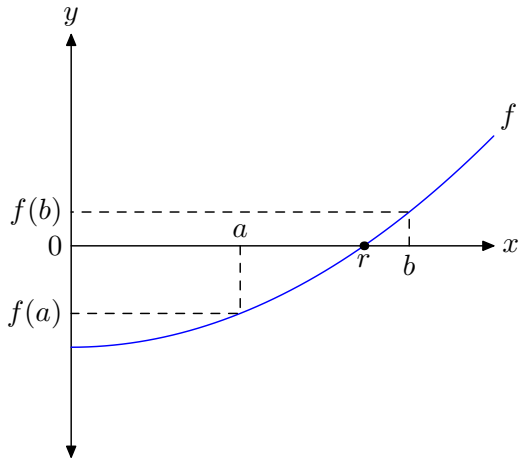


Figura 2: Método da Bisecção

Defina uma função para computar o valor da função $x - \cos(x)$, e então escreva um programa para encontrar um zero para esta função no intervalo $[0, \pi/2]$.

O programa deve ser implementado usando as funções com as seguintes interfaces:

```
void ledados(float* x, float* y)
float bissec(float x, float y)
void imprimeresultado(float x, float y, float raiz)
```

A função `bissec` recebe os valores do intervalo $[x, y]$ e calcula um zero da função $f(x) = x - cos(x)$ no intervalo $[x, y]$. Assumimos que $f(x) * f(y) < 0.0$.

13. (`zeronewton.c`) Outro método usado para localizar um zero de uma função é o método de Newton. Este método consiste de tomar uma aproximação inicial $x_0$ e construir uma linha tangente ao gráfico de $f$ neste ponto. O ponto $x_1$ onde esta tangente corta o eixo $x$ é tomado como uma segunda aproximação para o zero. Então outra linha tangente é construída em $x_1$, e o ponto $x_2$ onde esta tangente corta o eixo $x$ é a próxima aproximação. A Figura 3 mostra este processo.

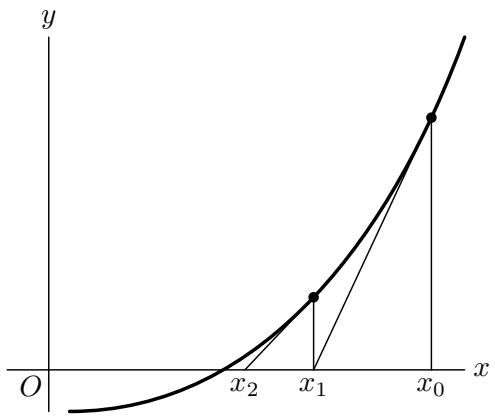


Figura 3: Método de Newton

Se $x$ é uma aproximação para o zero de $f$, a fórmula para obter a próxima aproximação pelo método de Newton é

Nova aproximação $= x - \frac{f(x)}{f'(x)}$

onde $f'$ é a derivada de $f$. Usando uma função para definir a função $f$ e sua derivada $f'$, escreva um programa para localizar um zero de $f$ usando o método de Newton. O processo deve terminar quando um valor de $f(x)$ é suficientemente pequeno em valor absoluto ou quando o número de iterações exceda algum limite superior. Imprima a sequência de aproximações sucessivas.

O programa deve ser implementado usando as funções com as seguintes interfaces:

```
void ledados(float* x, float* y)
float newton(float x, float y)
void imprimeresultado(float x)
```

A função newton recebe o valor da primeira aproximação $x_0$ e o valor da aproximação e calcula um zero da função $f(x) = x - cos(x)$. Assumimos que a função $f(x)$ seja diferenciável e tenha pelo menos um zero.

**Exercícios de Simulação**

14. Uma moeda é repetidamente lançada e um prêmio de $2n$ é pago, onde $n$ é o número da jogada na qual a primeira cara aparece. Por exemplo `CoCoCa` para R$ 8, `CoCa` paga R$ 4 e `Ca` paga R$ 2. Escreva um programa para simular 100 jogadas deste jogo, e imprimir a média do prêmio para estas jogadas.

15. Suponha que um jogador faça uma aposta de R$ 5 no seguinte jogo. Um par de dados é lançado, e se o resultado for ímpar, o apostador perde sua aposta. Se o resultado for par, uma carta é retirada de um baralho padrão de 52 cartas. Se a carta retirada for um Ás, 3, 5, 7 ou 9, o jogador ganha o valor da carta; caso contrário ele perde. Qual é a média de ganho para este jogo? Escreva um programa para simular o jogo.

16. Epaminondas Passarinho, campista central (CF) do time de beisebol Aves Pantaneiras, tem as seguintes porcentagens como batedor.

| | |
|---|---|
| Out | 63.4% |
| Walk | 10.3% |
| Single | 19.0% |
| Double | 4.9% |
| Triple | 1.1% |
| Home run | 1.3% |

Escreva um programa para simular 1000 vezes uma batida para Epaminondas, contando o número de outs, walks, singles e assim por diante, e calcule sua média de rebatidas ((número de batidas)/(1000 - número de walks)).

17. `(drunk.c)` O clássico problema da caminhada do bêbado é o seguinte: Numa rua de oito quadras, a casa de um bêbado está na quadra 8, e um bar na quadra 1. Nosso pobre bêbado começa na quadra $n$, $1 < n < 8$, e vagueia aleatoriamente, uma quadra de cada vez, em direção de sua casa ou em direção do bar. Em cada esquina, ele se move em direção do bar com uma determinada probabilidade, digamos 2/3, e em direção de sua casa com uma determinada probabilidade, digamos 1/3. Chegando a sua casa ou ao bar, ele permanece lá. Escreva um programa para simular 500 caminhadas na qual ele começa na quadra 2, outras 500 na qual ele começa na quadra 3, e assim por diante até a quadra 7. Para cada ponto inicial, calcule e imprima a porcentagem das vezes que ele termina em casa e a média do número de quadras que ele anda em cada caminhada.

O programa deve ser implementado usando as funções com as seguintes interfaces:

```
void calculatrajeto(int quadrai, float* bar, float* casa,
                    float* media)
void imprimeresultado(int quadrai, float bar, float casa, float media)
```

A função `calculatrajeto` recebe os valores de `quadrai`, e as referências de `bar`, `casa` e `media` que serão calculados na função. A função `imprimeresultado` imprime o valor da `quadrai`, e as probabilidades da caminhada finalizar no `bar`, na `casa` e a `media` de quadras caminhadas. Para a geração dos números pseudo aleatórios utilize a semente 997.

```
srand(997);
```

Exemplo de entrada:

```
drunk.in
```

A entrada para o programa `(drunk.c)` é vazia.

Exemplo de saída:

```
raiz.sol
2 0.994000 0.006000 2.766000
3 0.984000 0.016000 5.472000
4 0.932000 0.068000 7.628000
5 0.902000 0.098000 9.770000
6 0.768000 0.232000 10.000000
7 0.518000 0.482000 7.914000
```

18. (`buffon.c`) O famoso Problema da Agulha de Buffon é o seguinte: Uma tabela é delimitada com linhas paralelas equidistantes, uma agulha com comprimento igual a distância entre essas linhas é jogada aleatoriamente na tabela. Escreva um programa que leia a distância $l$ entre as linhas (comprimento da agulha) e simule este experimento. O seu programa deve estimar a probabilidade $p$ da agulha cruzar uma dessas linhas. Imprima os valores de $p$ e $2/p$. (O valor de $2/p$ deve ser igual a uma bem conhecida constante. Que constante é essa?)

    O site Buffon tem um applet Java que simula o problema.

19. Um método não muito usual para a aproximação da área sob uma curva é a técnica de Monte Carlo. Como ilustrado na Figura 4, considere um retângulo com base $[a, b]$ e altura $m$, onde $f(x) \leq m$ para todos $x$ em $[a, b]$. Imagine o lançamento de q dardos no retângulo $ABCD$ e contando o número total $p$ que acertam a região sombreada. Para um número grande de lançamentos, temos que

$$\frac{p}{q} \leq \frac{\text{área da região sombreada}}{\text{área do retângulo } ABCD}$$

    Escreva um programa para calcular áreas usando o método de Monte Carlo. Para simular o lançamento de um dardo, gere dois números aleatórios, $X$ para $[a, b]$ e $Y$ para $[0, m]$, e considere o ponto $(X, Y)$ onde o dardo acerta.

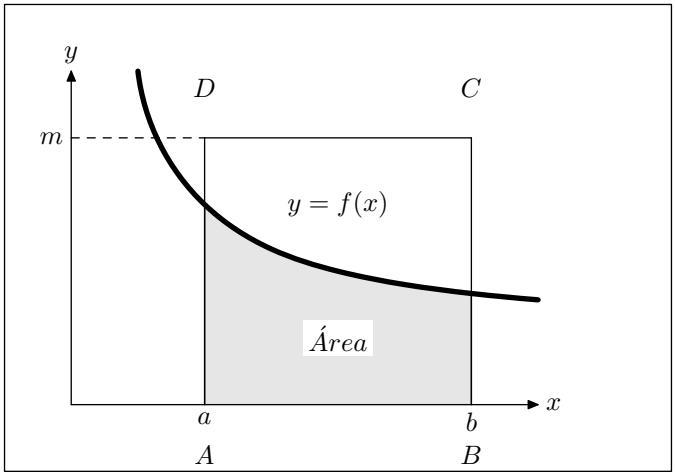


Figura 4: Área sob a curva - Monte Carlo

20. A Figura 5 mostra uma seção de corte de um muro de chumbo para proteção de um reator nuclear. Um nêutron entra no muro no ponto E e então segue um caminho aleatório movimentando-se para frente, para trás, a direita e a esquerda, em saltos

de uma unidade. Uma mudança de direção é interpretada como uma colisão com um átomo de chumbo. Suponha que após 10 dessas colisões a energia do nêutron é dissipada e que ele morre dentro da proteção de chumbo, providenciando que ele não volte para dentro do reator ou atravesse o muro de chumbo. Escreva um programa que leia a espessura do muro de chumbo e simule 100 nêutrons entrando em qualquer ponto nesta proteção, e calcule quantos deles sairão fora do reator. O muro de proteção é circular e os nêutrons só conseguem sair para fora do muro de proteção, voltar para dentro do reator ou morrer dentro do muro da proteção.

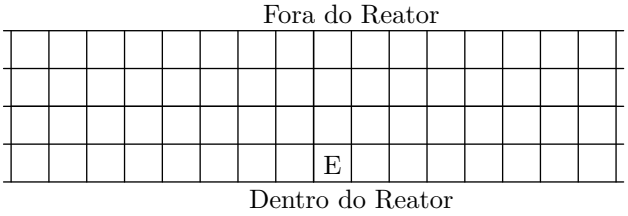


Figura 5: Radiação