Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Programming Language I • IMD0030

◁ Exercícios de "Aquecimento" em Programação ▷

9 de fevereiro de 2017

1. Escreva um programa em C++ chamado `negativo5.cpp` que lê 5 valores inteiros, um de cada vez, conta quantos destes valores são negativos e imprime esta informação.

2. Escreva um programa em C++ chamado `intervalos.cpp` que lê um número não conhecido de valores, um de cada vez, e conta quantos deles estão em cada um dos intervalos $[0, 25)$, $[25, 50)$, $[50, 75)$ e $[75, 100]$.

   Para encerrar a entrada de dados o usuário deve pressionar `<Ctrl+d>`. Para ler valores do terminal até o usuário digitar `<Ctrl+d>` você pode utilizar o seguinte trecho de código:

   ```
   int x;
   ...
   cout << "Entre com valores inteiros (Ctrl+d p/ encerrar): " << endl;
   while( cin >> x ) {
       // Realização da contagem de ocorrências nos intervalos
       ...
   }
   // Exibir contagem para os intervalos solicitados.
   ```

   Após encerrada a entrada de dados, o programa deve imprimir a porcentagem de números para cada um dos quatro intervalos indicados.

3. Escreva um programa em C++ chamado `soma_pares.cpp` que lê um número não determinado de pares "m n" (sem as aspas), todos inteiros e positivos, um par de cada vez, calcula e escreve a soma dos n primeiros inteiros consecutivos à partir de m (inclusive). Para encerrar a entrada de dados você deve utilizar `<Ctrl+d>`. Por exemplo, se uma entrada for "3 5" o progama deve calcular a soma dos 5 primeiros inteiros a partir de 3 (inclusive), ou seja, $3 + 4 + 5 + 6 + 7 = 25$ e imprimir como resultado 25.

4. Implemente um programa em C++ chamado `fib_up_n.cpp` que recebe um valor inteiro positivo `L` e imprime os termos da série de Fibonacci **inferiores** a `L`.

   A sequência de Fibonacci define-se como tendo os dois primeiros termos iguais a 1 e cada termo seguinte é à soma dos dois termos imediatamente anteriores. Desta forma se fosse fornecido ao programa uma entrada $\texttt{L} = 15$ o mesmo deveria produzir a seguinte sequência de termos da série: $1\ 1\ 2\ 3\ 5\ 8\ 13$.

5. Escreva um programa em C++ chamado `menor_elemento.cpp` que lê 20 números reais, os armazena em um arranjo unidimensional (vetor) `Vet` e o imprime na tela. A seguir, o programa

deve encontrar o menor elemento e a sua posição no vetor `Vet` e escrever na saída padrão qual é o menor elemento e que posição ele ocupa no vetor.

6. Escreva um programa em C++ chamado `troca_interna.cpp` que lê 20 inteiros, os armazena em um arranjo unidimensional (vetor) `A` e o imprime na tela. A seguir, o programa deve trocar o conteúdo do último elemento de `A` com o 1º, do penúltimo com o 2º, do ante-penúltimo com o 3º e assim pode diante até que todos os elementos tenham sido trocados de lugar *apenas uma vez*. Por fim, o programa deve imprimir o vetor modificado.

7. Escreva um programa em C++ chamado `troca_seguintes_vet.cpp` que lê 20 inteiros, os armazena em um arranjo unidimensional (vetor) `B` e o imprime na tela. A seguir, o programa deve trocar o conteúdo dos elementos de `B` de ordem ímpar com os de ordem par imediatamente seguintes e imprimir o vetor modificado.

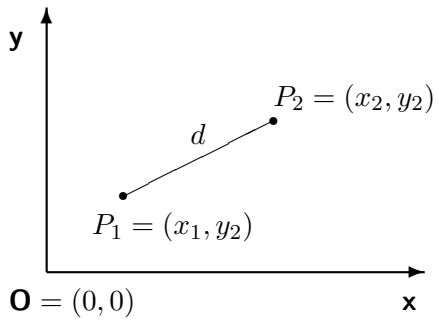
8. Escreva um programa em C++ chamado `compacta_vet.cpp` que lê 20 inteiros, os armazena em um arranjo unidimensional (vetor) `C` e o imprime na tela. A seguir, o programa deve "compactar" o vetor `C`, retirando dele todos os valores nulos ou negativos e imprimir o vetor modificado. Note que, dependendo dos dados de entrada, o vetor pode ter seu tamanho *lógico* reduzido; em outras palavras, é necessário criar um índice `length` que inicialmente vale 20, mas depois do processamento ele pode valer menos que 20, dependendo de quantos elementos negativos foram "eliminados" no processo. Considere o exemplo abaixo com apenas 10 elementos:

C:	-2	-8	2	7	-3	10	1	-1	-3	7
	0	1	2	3	4	5	6	7	8	9

depois de compactado fica com tamanho "lógico" $= 5$

C:	2	7	10	1	7	10	1	-1	-3	7
	0	1	2	3	4	5	6	7	8	9

9. Implemente um programa em C++ chamado `dist_pts.cpp` que calcula e imprime a distância Euclidiana entre dois pontos do $\mathbb{R}^2$, $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$, determinados por suas coordenadas Cartesianas (veja figura abaixo). Para tanto o programa deverá ler as coordenadas x e y de cada um dos dois pontos. A fórmula para o cálculo da distância d entre dois pontos P_1 e P_2 é dada por: $d(\overline{P_1P_2}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.



10. Implemente um programa denominado `equacao.cpp` que invoca a função `raizes`, que por sua vez 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.

Vale salientar que:

1. Se a equação possuir raízes reais e distintas, `x1` deve armazenar a raiz menor e `x2` a outra raiz. Neste caso a função deve retornar o valor 2, indicando que existem duas raízes.
2. Se a equação possuir possuir raízes reais e iguais `x1` $=$ `x2` e a função deve retornar 1.
3. Se a equação não possuir raízes reais então a função deve fazer `x1` $= 0$ e `x2` $= 0$ e o retorno da função deve ser 0.
4. A função `sqrt` definida na biblioteca padrão da linguagem deve ser usada para calcular a raiz quadrada de números. Para tanto basta incluir a biblioteca `<cmath>` onde está declarado o protótipo da função: `double sqrt(double n)`

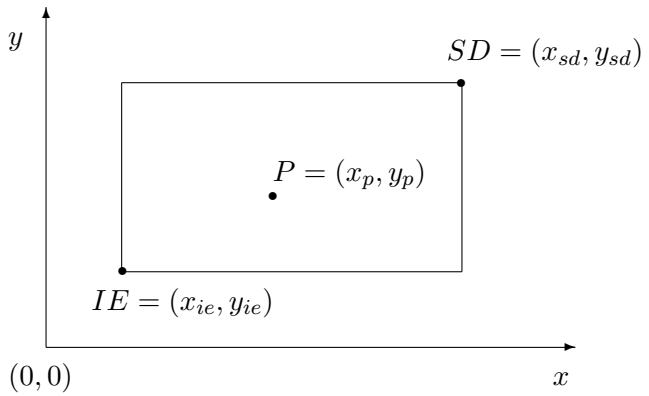
11. Escreva um programa em C++ chamado `vet5.cpp` que recebe um conjunto de 30 valores inteiros e os coloca em 2 vetores, `A` e `B`, conforme forem pares ou ímpares. Os vetores `A` e `B` deverão ter 5 posições de armazenamento. Quando um dos dois vetores fica cheio, o programa deve imprimi-lo. Terminada a entrada de dados, o programa deve imprimir o conteúdo dos dois vetores. Note que cada vetor pode ser preenchido tantas vezes quantas forem necessárias.

12. Implemente um programa em C++ chamado `hanoi.cpp` que resolve, recursivamente, o problema das Torres de Hanoi. O problema das Torres de Hanoi consiste em 3 pinos - A (origem), B (destino) e C (auxiliar) - e n discos de diâmetros diferentes. Inicialmente, todos os discos se encontram empilhados no pino origem (A), em ordem decrescente de tamanho, de baixo para cima. O objetivo é empilhar todos os discos no pino-destino (B), atendendo às seguintes restrições: (i) apenas um disco pode ser movido de cada vez, e; (ii) qualquer disco não pode ser jamais colocado sobre outro de tamanho menor.

O programa deve receber, via linha de comando, um valor inteiro positivo correspondente ao número de discos a ser resolvido e deve exibir a sequência de movimentos para achar a resposta, seguido do número de movimentos feito no total. Segue abaixo um exemplo de execução desse programa:

```
$ ./hanoi 3
Mova disco de A para B
Mova disco de A para C
Mova disco de B para C
Mova disco de A para B
Mova disco de C para A
Mova disco de C para B
Mova disco de A para B
Foram necessários 7 movimentos.
```

13. Implemente um programa em C++ chamado `ponto_em_retangulo.cpp` que verifica se um ponto $P = (x_p, y_p)$, determinado por suas coordenadas Cartesianas, está localizado *dentro*, *na borda* ou *fora* de um retângulo definho por dois pontos (também determinados por suas coordenadas Cartesianas): o canto inferior esquerdo $IE = (x_{ie}, y_{ie})$ e o canto superior direito $SD = (x_{sd}, y_{sd})$.



O programa deve receber três pares de valores x e y. Os dois primeiros pares correspondentes aos dois pontos, R_1 e R_2, que definem um retângulo; enquanto que o último par define o ponto de teste P. Note que R_1 e R_2 podem não corresponder diretamente ao canto inferior esquerdo e canto superior direito nesta ordem, podendo seus valores estarem trocados. Portanto o programa deve, primeiramente, assegurar-se de que R_1 corresponda a IE e R_2 corresponda a SD, trocando os valores de suas coordenadas se for o caso. Além disso o programa também deve assegurar-se de que R_1 e R_2 de fato definem um retângulo válido (isto é, $R_1 \neq R_2$). A seguir o programa deve realizar testes e indicar se o ponto P está *dentro*, *na borda* ou *fora* do retângulo, imprimindo uma mensagem para cada situação.

14. Implemente um programa denominado `q6.cpp` que cria um vetor de inteiros (pode ser estático ou fornecido pelo usuário) e invoca a função `maiores`. Esta função recebe como parâmetro o 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. O protótipo da função é:

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

15. Desenvolva funções para implementar as seguintes ações:

    1. `Swap`: Trocar o conteúdo de duas variáveis passadas *por referência*.
    2. `Ordena3`: Recebe como parâmetros três números inteiros e um *flag* `ordem`, e os coloca em ordem crescente se `ordem`$= V$, ou em ordem decrescente se `ordem`$= F$ (usar passagem de parâmetro *por referência* e a função `Swap()`).
    3. `EhPrimo`: Verifica se um número recebido como parâmetro é primo[1].
    4. `EhPar`: Retorna verdadeiro (`V`) se um número recebido como parâmetro é par, retornando falso (`F`) caso contrário.
    5. `EhAmigo`: Retorna verdadeiro (`V`) se os dois números recebidos como parâmetros são amigos[2], falso (`F`) caso contrário.
    6. `mdc`: Retorna o Máximo Divisor Comum[3] de 3 números recebidos como parâmetros.
    7. `mmc`: Retorna o Mínimo Múltiplo Comum[4] de 3 números recebidos como parâmetros.
    8. `Fatorial`: Retorna o fatorial do número recebido como parâmetro.

16. Implemente um programa em C++ denominado `ordena_insert.cpp` que lê, para um vetor `V` de 30 posições, vinte valores inteiros que ocuparão as 20 primeiras posições de `V`. Ordene, a seguir, os elementos de `V` em ordem não decrescente. Leia, a seguir, 10 valores inteiros, um por vez, e insira-os nas posições adequadas de `V`, de forma que `V` continue ordenado em ordem não decrescente. Ao final imprima o conteúdo de `V`.

17. Faça um programa chamado `deslocamentos.cpp` que recebe quatro números inteiros, `n1`, `n2`, `n3` e `n4`, e um certo número de deslocamentos, `d`, que esses números devem sofrer em relação a sequência original de entrada de dados. Se `d=0` a sequência de saída é a mesma da entrada; se `d>0` você deve deslocar os números para a 'direita' `d` vezes, imaginando que os números formam um círculo (i.e. após o n-ésimo termo segue-se o 1°); se `d<0` o deslocamento deve ser feito para a 'esquerda'.

    Por exemplo, se a entrada for $n1 = 5$, $n2 = -2$, $n3 = 7$, $n4 = 45$ e $d = 3$ o resultado final seria $(-2, 7, 45, 5)$, ou seja, os números foram deslocados para a direita três vezes: $(5, -2, 7, 45) \rightarrow (45, 5, -2, 7) \rightarrow (7, 45, 5, -2) \rightarrow$ **(-2,7,45,5)**.

    Utilize uma função denominada `ShiftN(n1,n2,n3,n4,d)` que recebe os números como parâmetros *por referência* e aplica o deslocamento `d` sobre eles. Tente fazer esta função de forma mais otimizado possível, evitando deslocamentos desnecessários (dica: utilize o operador `resto` ou `%` em C++).

[1] Um número natural maior do que 1 cujos únicos divisores naturais são 1 e o próprio número.

[2] Dois números são *amigos* se cada um deles é igual a soma dos divisores próprios do outro (os *divisores próprios* de um número positivo n são todos os divisores inteiros positivos de n exceto o próprio n).

[3] Maior número inteiro encontrado que seja *fator* dos números sobre o qual deseja-se achar o mdc.

[4] $mmc(a, b) = \frac{ab}{mdc(a,b)}$.

18. Faça uma função em C++ chamada `inverteVetor` que recebe um vetor de inteiros como parâmetro e inverte a ordem dos seus elementos, alterando o vetor original passado como argumento. O parâmetro formal da função correspondente ao vetor a ser invertido deve ser um **ponteiro simples**. Utilize aritmética de ponteiros ao invés de indexação para percorrer o vetor. Implemente sua função de tal forma que não seja necessário o uso de um vetor auxiliar e nem percorrer todos os elementos do vetor original. A função `inverteVetor` deve obedecer o seguinte protótipo:

    ```
    void inverteVetor(int *piVet, const int tam);
    ```

    onde `piVet` é o ponteiro para o vetor a ser invertido e `tam` é o tamanho do mesmo.

19. Faça uma função em C++ chamada `intercalaVetores` que recebe como parâmetro dois vetores, `vetA` e `vetB`, de caracteres (possivelmente de tamanhos diferentes) e os combina em um só vetor. Para gerar o vetor combinado deve-se intercalar elementos dos dois vetores da seguinte forma: para o `vetA` deve-se iniciar com seu primeiro elemento, avançando sequencialmente pelo vetor até o seu último elemento; para o `vetB` deve-se iniciar pelo último elemento, retrocedendo sequencialmente pelo vetor até o seu primeiro elemento.

    Note que a função deve alocar dinamicamente uma quantidade de memória do tamanho exato para conter a combinação resultante dos dois vetores. A função deve utilizar aritmética de ponteiros para percorrer os dois vetores passados como argumento.

    A função `intercalaVetores` deve obedecer o seguinte protótipo:

    ```
    char * intercalaVetores( const char * vetA, int tamA,
                             const char * vetB, int tamB );
    ```

    onde `vetA` é o ponteiro para o primeiro vetor e `tamA` seu tamanho; `vetB` é o ponteiro para o segundo vetor e `tamB` seu tamanho. A função deve retornar um ponteiro para o vetor contendo a combinação dos dois vetores passados como parâmetro.

20. Desenvolva duas funções, `NesimoFib(n)` e `FibMenorL(L)`, que imprimem, respectivamente, a sequência de Fibonacci até seu n-ésimo termo e os m primeiros termos da série de Fibonacci que são *menores* que `L`. Lembre-se que a série de Fibonacci é dada por $\{1, 1, 2, 3, 5, 8, 13, 21, \dots\}$, ou seja, os primeiros dois termos da série são 1 e os próximos termos são calculados como a soma dos dois termos precedentes imediatos.

    Desenvolva também uma função denominada de `PiramideFib(h)` que recebe como parâmetro h indicando a altura da pirâmide de Fibonacci e a imprime da tela (utilize as funções anteriores). A pirâmide de Fibonacci para $h = 7$ deve corresponder a:

    ```
          1
        1 1
      2 1 1
    ```

```
      3 2 1 1
    5 3 2 1 1
  8 5 3 2 1 1
13 8 5 3 2 1 1
  8 5 3 2 1 1
    5 3 2 1 1
      3 2 1 1
        2 1 1
          1 1
            1
```

Note que a pirâmide esta "deitada" para a esquerda.

21. Implemente um programa chamado `avalia_poli.cpp` que recebe como entrada de dados inteiros correspondente a um polinômio e um valor real para o qual o polinômio deve ser avaliado. O polinômio poderá ter, no máximo, 4 *termos* ou *monômios*.

    Por exemplo, a seguinte entrada de dados:

    ```
    4 2 -2 1 5 0 0 0 5.0
    ```

    corresponde ao polinômio de grau 2, $4x^2 - 2x + 5$, o qual deve ser avaliado para o valor $x = 5.0$, resultando $4(5.0)^2 - 2(5.0) + 5 = 95.0$.

    Após a leitura de um polinômio e de um valor a ser avaliado o programa deve invocar a função `avalia` que deve receber os 4 termos do polinômio e o valor a ser avaliado como parâmetros e retornar o valor da avaliação. O resultado da função deve ser exibido na tela, juntamente com uma representação do polinômio, da seguinte forma:

    ```
    f(x)=4x^2 - 2x + 5, f(5.0) = 95.0
    ```

    Nesta representação deve haver apenas um e apenas um espaço em branco ' ' antes e depois dos sinais '-' e '+'.

22. Faça uma função em C++ chamada `uniaoIntersecao` que recebe como parâmetro dois vetores, `vetA` e `vetB`, de inteiros (possivelmente de tamanhos diferentes) e os combina em dois vetores de saída, `vetUni` e `vetInt`. Para gerar o vetor `vetUni` a função deve realizar a *união* entre os elementos de `vetA` e `vetB`. Para gerar o vetor `vetInt` a função deve realizar a *intersecção* entre os elementos de `vetA` e `vetB`.

    Note que a função deve alocar dinamicamente os dois vetores de saída, i.e., `vetUni` e `vetInt`, de tal forma que os mesmos tenham o tamanho **exato** do resultado de suas respectivas operações de combinação. A função deve utilizar aritmética de ponteiros para percorrer os dois vetores de entrada, `vetA` e `vetB`, passados como argumento.

    A função `uniaoIntersecao` poderia ter o seguinte protótipo:

```
void uniaoIntersecao( int *vetA, int tamA, int *vetB, int tamB,
                      int **vetUni, int *tamUni,
                      int **vetInt, int *tamInt );
```

onde `vetA` e `vetB` são os vetores de entrada com seus respectivos tamanhos, `tamA` e `tamB`; `vetUni` está associado ao vetor que conterá a união dos vetores de entrada, cujo tamanho será indicado em `tamUni`; e `vetInt` está associado ao vetor que conterá a intersecção dos vetores de entrada, cujo tamanho será indicado em `tamInt`. Modifique o protótipo acima introduzindo o qualificador de tipo `const` de tal forma a restringir ao máximo o acesso aos dados externos mas que ainda permita a função executar o seu propósito original.

Implemente um programa em C++ chamado `test_string.cpp` onde você deverá implementar as funções solicitadas nas questões 23 à 26. Não esqueça de declarar o protótipo de cada função no início do programa. Basicamente o que o programa deve fazer é criar uma série de *strings* e aplicar cada uma das funções de forma a testar sua funcionalidade e se a mesma funciona como deveria.

Algumas observações devem ser feitas com relação a funções que criam e manipulam *strings*:

- Geralmente a *string* passada como parâmetro não deve ser alterada, e;
- Novas *strings* deve ser criada com alocação dinâmica (uso de `new` e `delete`[5]).

23. Implemente uma função que receba uma *string* e um número inteiro `n` como parâmetros, e retorne uma nova *string* com os `n` primeiros caracteres da *string* passada como parâmetro. Por exemplo, recebendo como parâmetros a *string "Gosto de programar em C++"* e o número 5, essa função deve alocar dinamicamente uma nova *string* contendo a sequência de caracteres *"Gosto"*. Essa função deve obedecer o protótipo:

    ```
    char * prefix ( const char * str, int n );
    ```

    Obs.: Teste se `n` é sempre menor do que o tamanho da *string* (que pode ser recuperado com a função `strlen`); se `n` for maior ou igual ao tamanho da *string* função deve simplesmente retornar uma cópia da *string* original.

24. Implemente uma função que receba uma *string* como parâmetro e retorne uma nova *string* que é o reverso da *string* original. Por exemplo, recebendo como parâmetros a *string "Sabedoria e honra"* a função deve retornar uma nova *string* contendo *"arnoh e airodebaS"*. Essa função deve obedecer o protótipo:

    ```
    char * reverse ( const char * str );
    ```

[5] Veja a observação no final desse documento.

25. Implemente uma função que receba uma *string* como parâmetro e retorne uma nova *string* com os caracteres minúsculos trocados para maiúsculas e vice-versa. Caracteres que não forem letras devem ser copiados sem alteração para a nova *string*. Por exemplo, se for passado como parâmetro a *string "UFRN - DIMAp - Dim0426"*, essa função deve retornar uma nova *string* contendo a sequência *"ufrn - dimaP - dIM0426"*. Essa função deve obedecer o protótipo:

```
char * invertCase ( const char * str );
```

26. Implemente uma função que receba uma *string* como parâmetro e retorne uma nova *string* com as letras da *string* original substituídas por suas sucessoras no alfabeto. Por exemplo, recebendo como parâmetro a *string "Casa"*, essa função retornaria a *string "Dbtb"*. Essa função deve obedecer o protótipo:

```
char* shiftString ( const char * str );
```

Obs.: A letra 'z' deve ser substituída pela letra 'a' (e 'Z' por 'A'). Caracteres que não forem letras devem ser copiados para a nova *string* sem sofrer alteração. A *string* passada como parâmetro não pode ser alterada.

27. Implemente um programa em C++ chamado `busca_palindromo.cpp` que lê uma *string* do usuário e verifica se a mesma é um *palíndromo*. Um palíndromo é uma palavra, frase, número ou qualquer outra sequência de unidades (como uma cadeia de ADN) que tenha a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita (o ajustamento de espaços entre letras é geralmente permitido) [6].

Exemplos de palavras: anilina, arara, mirim, radar, rotor, osso, ovo, iriri, mussum.

Exemplos de frases: *"A rara arara"*, *"Socorram-me, subi no ônibus em Marrocos"*, *"A mala nada na lama"*.

Identificar palíndromos em palavras é relativamente fácil. A maior dificuldade será identificar frases que são palíndromos. Para tanto o programa deverá eliminar todos os espaços em brancos e separadores como hífen, vírgulas, ponto-e-vírgula, etc.

Obs.: Considere que serão fornecidas apenas *strings* sem acentos gráficos como á ou é; crie funções parar eliminar os espaços em brancos e separadores de forma a facilitar a busca de palíndromos.

28. Escreva um programa que lê valores reais para uma matriz `M[5][5]` e calcula e imprime as seguintes somas:

    a) da linha 4 de `M`

    b) da coluna 2 de `M`

[6] Fonte: http://pt.wikipedia.org.

c) da diagonal principal de `M`

d) da diagonal secundária de `M`

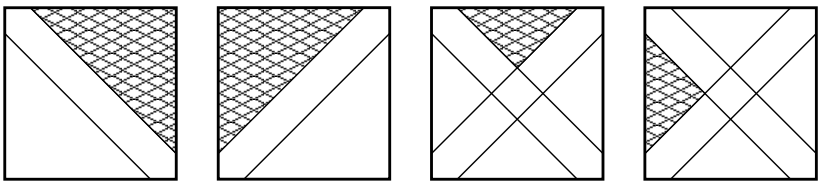
e) de todos os elementos da matriz

29. Implemente um programa em C++ chamado `preenche_matriz.cpp` que lê um inteiro positivos n e cria uma matriz quadrada n x n que deverá ser preenchida automaticamente com inteiros positivos de 1 até n^2, dando prioridade a linhas ao invés de colunas (*row-major*). Por exemplo, uma matriz 3 x 3 deverá ser preenchida da seguinte forma:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Faça o seu programa imprimir na tela a matriz assim preenchida.

30. Escreva um programa em C++ que lê uma matriz `M[6][6]`, calcula as somas das pares hachuriadas, imprime o conteúdo da matriz `M` e as somas calculadas.



31. Implemente uma função que indique se uma matriz quadrada de números inteiros é uma matriz identidade ou não. A função deve retornar 1 se a matriz for uma matriz identidade, e 0 caso contrário (matriz identidade é aquela em que a diagonal principal contém apenas $1s$ e o resto da matriz contém zeros). A função recebe como parâmetro a matriz de inteiros, usando a representação de matrizes através de **ponteiro simples**, e um inteiro `n`, indicando a dimensão da matriz. Essa função deve obedecer o protótipo:

```
int ehMatrizIdentidade ( const int * mat, int n );
```

32. Implemente uma função que indique se uma matriz quadrada de números inteiros é uma matriz triangular superior ou não. A função deve retornar 1 se a matriz for uma matriz triangular superior, e 0 caso contrário. A função recebe como parâmetros a matriz de inteiros, usando a representação de matrizes através de **ponteiro simples**, e um inteiro n, indicando a dimensão da matriz. Essa função deve obedecer o protótipo:

```
int ehTrianguloSuperior ( const int * mat, int n );
```

33. Implemente uma função que retorne a soma dos elementos abaixo da diagonal de uma matriz quadrada de números de ponto flutuante (`float`), isto é, a soma de todos os elementos $a_{i,j}$

onde $i > j$. A função recebe como parâmetros a matriz de números de ponto flutuante, usando a representação de matrizes através de **ponteiro duplo**, e um inteiro n, indicando a dimensão da matriz. Essa função deve obedecer o protótipo:

```
float somaTrianguloInferior ( const float ** mat, int n );
```

34. Uma matriz que tem aproximadamente $2/3$ de seus elementos iguais a zero é denominada de *matriz esparsa*. Implemente um programa em C++ chamado `esparsa.cpp` que lê uma matriz esparsa de inteiros `M[10][10]` e forma uma matriz condensada, de apenas 3 colunas com os elementos não nulos da matriz `M`, de forma que:

    a) a primeira coluna contém o valor não nulo de `M`.

    b) a segunda coluna contém a linha de `M` onde foi encontrado o valor.

    c) a terceira coluna contém a coluna de `M` onde foi encontrado o valor.

    A seguir, imprima a matriz lida e a matriz condensada.

35. Na teoria dos Sistemas define-se como elemento *minimáx* de uma matriz, o menor elemento da linha em que se encontra o maior elemento da matriz. Implemente um programa em C++ chamado `minmax.cpp` que lê uma matriz `A[10][10]` e determina o elemento *minimáx* desta matriz, imprimindo a matriz `A` e a posição do elemento *minimáx*.

36. Considerando a estrutura

```
typedef struct _Ponto {
      int x;
      int y;
   } Ponto;
```

    para representar um ponto em uma grade 2D, implemente uma função que indique se um ponto p está localizado dentro ou fora de um retângulo. O retângulo é definido por seus vértices inferior esquerdo `v1` e superior direito `v2`. A função deve retornar 1 caso o ponto esteja localizado dentro do retângulo e 0 caso contrário. Essa função deve obedecer o protótipo:

```
int dentroRet (const Ponto* v1, const Ponto* v2, const Ponto* p);
```

    ou alternativamente use o seguinte protótipo:

```
int dentroRet (const Ponto &v1, const Ponto &v2,
               const Ponto &p);
```

    Obs.: Repare que os pontos `v1`, `v2` e `p` são passados como **referência constante** de forma que seus valores não podem ser alterados dentro da função. Isso é uma boa prática de programação, uma vez que para calcular se `p` está dentro do retângulo definido por `v1` e `v2` não é necessário modificar seus valores.

37. Considerando a estrutura da questão 36 para representar um ponto em uma grade 2D, implemente uma função que indique se um ponto `p` está localizado dentro ou fora de um círculo. O círculo é definido por seu centro `c` e seu raio `r`. A função deve retornar 1 caso o ponto esteja localizado dentro ou na borda do círculo e 0 (zero) caso contrário. Essa função deve obedecer o protótipo:

    ```
    int dentroCirc ( const Ponto *c, int raio, const Ponto *p);
    ```

38. Faça uma função denominada `PontoEmEsfera` que verifica se um ponto P especificado por suas coordenadas Cartesianas $P = (x, y, z)$ está dentro está *dentro*, *na superfície* ou *fora* de uma esfera de raio r e centro em $C = (x_c, y_c, z_c)$.

    Faça um programa chamado de `verifica_esferas.cpp` que lê, de um arquivo texto, uma sequência de definições de ponto e esfera (um conjunto por linha) e imprime a classificação do ponto (dentro, superfície ou fora) na saída padrão e em um arquivo denominado `saida_esferas.txt`. Segue abaixo um exemplo de formato de arquivo.

    ```
    1.5 2.5 1.0 3.0 3.0 3.0 4.0
    ...
    x y z xc yc zc r
    ```

    A fórmula da distância Euclidiana entre dois pontos P_1 e P_2 é:

    $$d(\overline{P_1P_2}) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}.$$

    A equação Cartesiana da esfera com centro (x_c, y_c, z_c) e raio r é dada por:

    $$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2.$$

    Observação: Pense como seria a solução do mesmo problema considerando uma *caixa* ao invés de uma esfera. Neste caso a caixa teria seus planos paralelos aos planos Cartesianos, mas seus comprimentos (altura, largura, profundidade) poderiam ser de tamanhos distintos. Um caixa pode ser especificada por apenas 2 vértices em faces opostas de forma a constituirem uma *diagonal principal* da caixa: canto inferior esquerdo na face da frente e canto superior direito na face de trás.

39. Considerando a estrutura (com `typedef`)

    ```
    struct _Vetor {
      float x;
      float y;
      float z;
    } Vetor;
    ```

para representar um vetor no ponto no $\mathbb{R}^3$, implemente uma função que calcule a soma de dois vetores. Essa função deve obedecer o protótipo:

```
void soma ( const Vetor *v1, const Vetor *v2, Vetor **res );
```

onde `v1` e `v2` são ponteiros para os vetores a serem somados pela função, e o parâmetro `res` é um ponteiro para um ponteiro para uma estrutura vetor onde o resultado da operação deve ser armazenado. Para tanto é necessário alocar dinamicamente a estrutura para receber o resultado da soma.

Além da função mencionada acima, implemente uma outra versão da soma que deve seguir o protótipo:

```
Vetor soma2 ( const Vetor *v1, const Vetor *v2 );
```

onde `v1` e `v2` são ponteiros para os vetores a serem somados pela função, e a função retorna uma estrutura do tipo `Vetor` que foi alocada dinamicamente pela função e contém o resultado da soma.

40. Considerando a estrutura da questão 39 para representar um vetor no $\mathbb{R}^3$, implemente uma função que calcule a subtração de dois vetores. Essa função deve obedecer o protótipo:

    ```
    void sub ( const Vetor *v1, const Vetor *v2, Vetor **res );
    ```

    onde `v1` e `v2` são ponteiros para os vetores a serem subtraídos pela função, e o parâmetro `res` deve ser alocado dinamicamente dentro da função e receber o resultado da operação de subtração $(v1 - v2)$.

41. Considerando a estrutura da questão 39 para representar um vetor no $\mathbb{R}^3$, implemente uma função que calcule o produto escalar de dois vetores. Essa função deve obedecer o protótipo:

    ```
    float dot ( const Vetor *v1, const Vetor *v2 );
    ```

    onde `v1` e `v2` são ponteiros para os vetores a serem multiplicados, e o resultado da operação $v1 \cdot v2$ deve ser retornado.

    O produto escalar entre dois vetores $\vec{X} = \{x_1, x_2, \ldots, x_n\}$ e $\vec{Y} = \{y_1, y_2, \ldots, y_n\}$ é $\vec{X} \cdot \vec{Y} = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + \cdots + x_n y_n$.

42. *Aritmética sobre os números naturais.*

    (i) Defina um módulo que implementa recursivamente o cálculo da multiplicação `MultSoma(m,n)` de dois números não-negativos `m` e `n` em termos de adições sucessivas.
    Dica: faça uso da equação de recorrência

    $$\text{MultSoma}(m, n) = \begin{cases} 0, \text{ se } n = 0 \\ \text{MultSoma}(m, n-1) + m, \text{ em caso contrário} \end{cases}$$

(ii) Defina um módulo que implementa recursivamente o cálculo da exponencial `ExpoMult(m,n)` de dois números não-negativos `m` e `n` em termos de multiplicações sucessivas. Qual é, neste caso, a equação de recorrência adequada para esta tarefa?

(iii) Faça uso dos dois módulos anteriores para definir o módulo `ExpoSoma`, que implementa o cálculo da exponencial de dois números não-negativos em termos de adições sucessivas.

(iv) Supondo a existência de um módulo que calcula a função `Sucessor`, que recebe um número natural e devolve o número que o segue segundo a ordem usual dos naturais, utilize tal módulo para implementar a função binária `Soma` sobre pares de naturais.

43. *Cálculo recursivo do Máximo Divisor Comum* (MDC).

(i) Dados dois números inteiros não nulos m e n, calcule seu MDC através da implementação da seguinte equação de recorrência:

$$\mathrm{MDC}(m,n) = \begin{cases} \mathrm{MDC}(m, m-n), \text{ se } m > n \\ \mathrm{MDC}(n-m, m), \text{ se } m < n \\ m, \text{ em caso contrário} \end{cases}$$

(Este procedimento é estendido e otimizado no algoritmo abaixo.)

(ii) **Algoritmo de Euclides** (cerca de 300 A.C.). Dados dois números naturais não-nulos m e n, tais que $0 \leq n < m$, calcule seu MDC através da implementação da seguinte equação de recorrência:

$$\mathrm{MDC}(m,n) = \begin{cases} m, \text{ se } n = 0 \\ \mathrm{MDC}(n, m \textbf{ resto } n), \text{ em caso contrário} \end{cases}$$

(iii) Dois números naturais positivos são ditos *co-primos* ou *primos entre si* caso eles não possuam fatores comuns para além da unidade. Use o módulo determinado por algum dos algoritmos acima para determinar se dois números `a` e `b` são primos entre si. Compare o número de chamadas recursivas efetuadas caso seja utilizado o módulo (i) com o número de chamadas geradas pelo módulo (ii).

44. Implemente cada um dos seguintes algoritmos de forma *imperativa* e de forma *recursiva*:

(i) Defina o módulo `MembroQ` que recebe um número `N` e um vetor `V` de números e devolve verdadeiro se `N` pertence a `V` e falso caso contrário.

(ii) Defina o módulo `Ocorrencias` que recebe um número `N` e um vetor `V` e conta quantas vezes `N` ocorre em `V`.

(iii) Defina um módulo que recebe um vetor `V` e um número `N` e devolve uma tripla `[A,B,C]` tal que `A` é o número de elementos de `V` maiores que `N`, `B` é o número de elementos de `V` iguais a N, e `C` é o número de elementos menores que `N`.

(iv) Defina um módulo `Prefixo` que recebe como argumento dois vetores `U` e `V` e devolve verdadeiro se `U` é prefixo de `V` e falso caso contrário.

(v) Defina uma sub-rotina `ProdEscalar` que recebe como argumento um vetor `V` e um escalar `X` e calcula o produto escalar de `V` por `X`, alterando diretamente o vetor `V`.

Dica. Acrescente a cada módulo alguns parâmetros de entrada convenientes para a realização da tarefa correspondente, tais como `dim` para a dimensão do vetor e `pos` para a posição do elemento em foco numa determinada execução do módulo em questão.

45. **Estudo de uma função.**

Sejam $X = [x_1, \ldots, x_n]$ e $Y = [y_1, \ldots, y_n]$ dois vetores tais que $f(x_i) = y_i$, para alguma função f. Assuma que o vetor X está ordenado.

(i) Defina a função `Máximo` que calcula o máximo da função f.

(ii) Defina a função `Monotonia` que devolve 1 se a função f é crescente, -1 se a função f é decrescente, e 0 caso contrário.

(ii) Defina a função `TVM` que calcula a Taxa de Variação Média de f em cada intervalo. Esta função deverá devolver um vetor de comprimento $n-1$ em que na i-ésima posição do vetor ocorre a TVM de f no intervalo $[x_i, x_{i+1}]$.

A **filtragem por convolução** é frequentemente empregada para reduzir os efeitos de ruído em imagens ou realçar detalhes de imagens desfocadas. A filtragem por convolução é uma forma de filtragem espacial que computa cada pixel de saída através da multiplicação de elementos de um *kernel* (i.e. uma matriz quadrada) com os pixels da imagem[7] original sendo processada. Portanto para efetuar o processo de filtragem precisamos de uma **imagem original**, um **kernel** e como resultado vamos gerar uma **imagem filtrada**.

O processo de filtragem consiste em mover o *kernel* sobre a *imagem original*, pixel por pixel, multiplicando-se a intensidade dos pixels sob o *kernel* pelo valor correspondente na *imagem original*. Desta forma os valores do *kernel* funcional como pesos, indicando a contribuição de cada pixel para o valor final. O resultado da soma dos valores assim multiplicados será o valor de intensidade do pixel da *imagem filtrada*, cuja posição corresponde à posição do pixel da *imagem original* logo abaixo do centro do *kernel*. A Figura 1 exemplifica o processo.

A cada nova interação devemos mover o *kernel* da esquerda para direita e de cima para baixo de forma a passar por todos os pixels da *imagem original*. Portanto a intensidade de um pixel na *imagem filtrada* depende da contribuição dos pixels vizinhos do pixel correspondente na *imagem original*. Note, contudo, que os pixels localizados nas bordas da *imagem original* não serão afetados pelo *kernel* e, portanto, poderão ser ignorados ou replicados na *imagem filtrada*.

A seleção dos pesos para os **elementos chave** do *kernel* determina a natureza da filtragem, tal como **filtro passa-baixo** e **filtro passa-alto**. Figura 2 apresenta alguns exemplos de *kernels*.

[7] Neste contexto considere uma *imagem* como sendo uma matriz de valores inteiros.

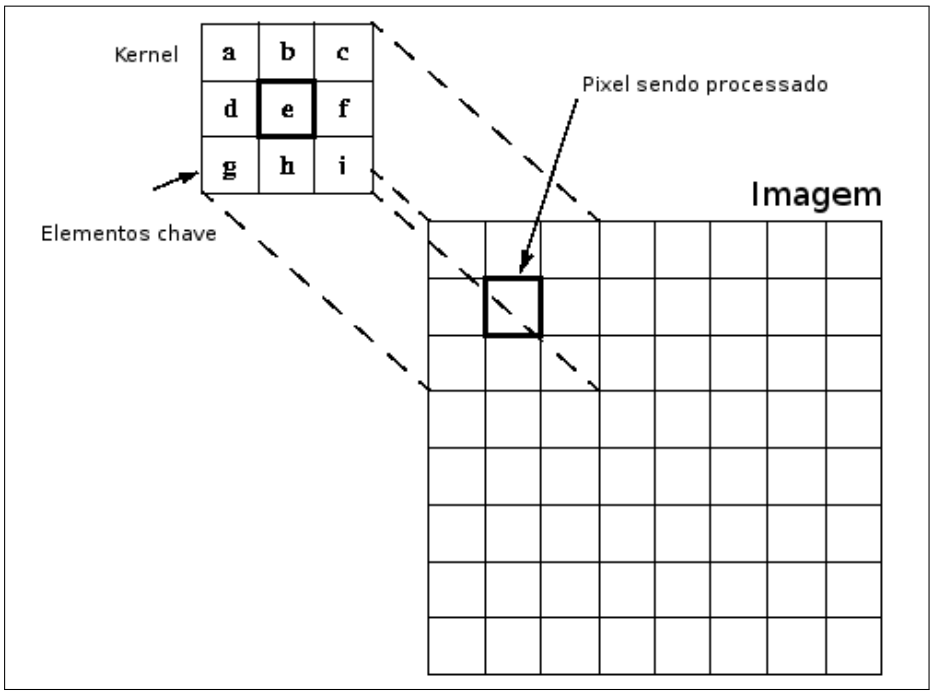


Figura 1: *Kernel* aplicado a imagem fonte.

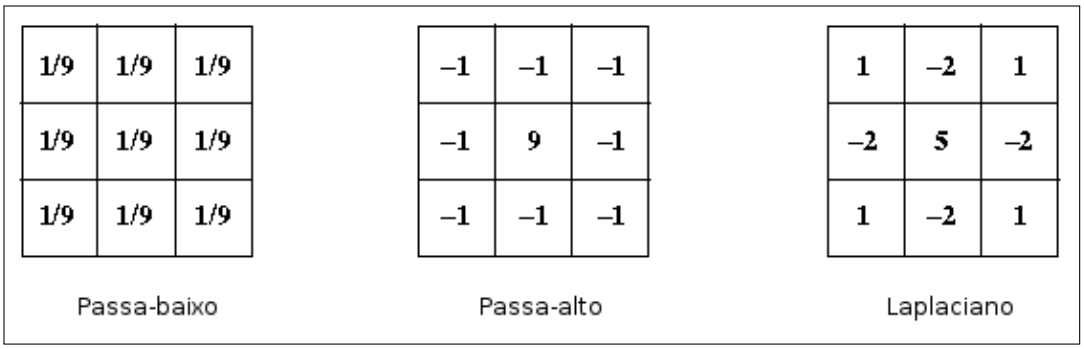


Figura 2: Exemplos de *kernel*.

46. Faça uma função chamada `filtroConvolucao` (protótipo livre) que deve receber como parâmetros uma matriz correspondente a uma *imagem original*, um matriz correspondente a um *kernel*, e um ponteiro duplo para uma matriz que receberá o resultado da filtragem: a *imagem filtrada*.

    A função deverá alocar dinamicamente a *imagem filtrada* e **não** deve permitir que a função altere os valores da *imagem original* nem do *kernel* (use o qualificador `const`). Para tanto assuma as seguintes simplificações: uma imagem é representada como uma matriz $n \times m$, na qual cada posição da matriz representa a intensidade de cinza de um pixel variando na faixa $[0; 255]$, para qual 0 representa a cor preta e 255 representa a cor branca; um *kernel* pode ser representado por uma matriz quadrada de valores reais.

    A *imagem original* deve ser passada como um **ponteiro simples** (use `const`). Portanto lembre-

se de especificar o número de linhas e colunas da *imagem original*. A função deverá retornar um ponteiro nulo, caso a alocação dinâmica falhe, ou o endereço da memória alocada para a *image filtrada*, caso contrário. Mas o mais importante é que a *image filtrada* deve ser retornada como um parâmetro da função, utilizando-se ponteiros duplos. O seu tamanho (linhas e colunas) também deve ser retornado via parâmetros (passagem por referência).

Observações

Alocações dinâmicas de memória devem ser realizadas com os operadores `new` e `delete` do C++. Além disso todas as entradas e saídas de dados devem ser feitas através de `cout` e `cin`. Para compilar os programas utilize o compilador g++ com a seguinte sintaxe:

```
$ g++ -Wall prog.cpp -o prog
```

∽ FIM ∽