# Lab. 8

# Vetores

**ECT2303 - T02 - 19.1**

Um vetor é uma estrutura de dados que pode armazenar multiplos valores do mesmo tipo. Nesse laboratório você aprenderá como criar e trabalhar com vetores em seus programas.

## 8.1 Declarando um vetor

- Para declarar um vetor, você precisa especificar o tipo (**`int`**, **`float`** ou **`double`**), bem como o **tamanho do vetor**.
- Para especificar o tamanho de um vetor, coloque o número de valores que o vetor pode armazenar dentro de colchetes. Por exemplo,

```
int a[10];
float b[100];
double c[50];
```

## 8.2 Inicializando um vetor

- Quando você atribui valores iniciais a um vetor, precisa delimitar os valores por abre e fecha chaves ({}).
- O comando a seguir inicializa um vetor de inteiros `a` com os valores 10, 15, 5, 20 e 25.

```
int a[5] = {10, 15, 5, 20, 25};
```

- Caso o número de inicializadores seja menor que o números de elementos do vetor, os elementos restantes são inicializados com zero. Por exemplo,

```
int b[5] = { 0 };
```

- Se você não especificar o tamanho do vetor, o compilador alocará memoria suficiente para conter somente os valores que você especificar. Por exemplo,

```
long longB[] = {1234567, 9847562, 8493041};
```

- Se você inicializar o vetor durante a compilação, seu programa será executado mais rapidamente.

## 8.3 Acessando os elementos do vetor

- Os valores armazenados no vetor são chamados de **elementos do vetor**.
- Para acessar o elemento do vetor você precisa especificar o nome do vetor e a posição do elemento dentro de colchetes. O primeiro elemento de um vetor está armazenado na posição 0 (zero). Por exemplo,

```
#include <iostream>
int main()
{
  int a[5] = {10, 15, 5, 20, 25};

  cout << "a[0] = " << a[0];
  cout << "a[1] = " << a[1];
  cout << "a[2] = " << a[2];
  cout << "a[3] = " << a[3];
  cout << "a[4] = " << a[4];

  return 0;
}
```

- Como alternativa, você pode usar uma variável para referenciar o elemento do vetor e uma estrutura de repetição `for` para acessar os elementos. Por exemplo,

```
#include <iostream>
#include <iomanip>

int main()
{
  int a[5] = {10, 15, 5, 20, 25};

  cout << "Elemento" << setw(13) << "Valor" << endl;
  for ( int i = 0; i < 5; i++ )
     cout << setw( 7 ) << i << setw( 13 ) << a[i] << endl;

  return 0;
}
```

## 8.4 Usando constantes para definir os vetores

- Usar variáveis constantes para especificar o tamanho de um vetor torna os programas mais flexíveis.

- Variáveis constantes devem ser inicializadas com uma expressão constante e não pode mais ser modificada. Por exemplo,

```
const int tamanhoVetor = 15;
```

- A variável constante **tamanhoVetor** pode ser usada para especificar o tamanho do vetor,

```
int a[ tamanhoVetor ];
```

- Quando os programas se tornam maiores, essa prática se torna muito útil. Por exemplo,

```
#include <iostream>
#include <iomanip>

int main()
{
  const int tamVetor = 10;
  int j, a[ tamVetor ];

  for ( j = 0; j < tamVetor; j++ )
    a[ i ] = 2 + 2*j;

  cout << "Elemento" << setw(13) << "Valor" << endl;
  for ( j = 0; j < tamVetor; j++)
    cout << setw( 7 ) << j << setw( 13 ) << a[ j ] << endl;

  return 0;
}
```

## 8.5 Passando vetores para funções

- Ao passar um vetor para uma função, é necessário passar o tamnaho do vetor, assim a função pode processar o número específico de elementos do vetor. Por exemplo,

```
tipo_func nome_func(tipo_vetor nome_vetor[], int tam_vetor)
{
      corpo da funcao
}
```

- O prótotipo da função é escrito como:

```
tipo_func nome_func(tipo_vetor [], int )
```

- Nas chamadas às funções, variáveis do tipo vetor são passadas como parâmetros utilizando apenas o seu nome

```
nome_func(nome_vetor, tam_vetor)
```

- Todo vetor passado para funções como parâmetro é passado por **referência**. Isto significa dizer que as alterações realizadas nos vetores dentro da função são visíveis fora da função, ou seja, todo vetor é um parâmetro de entrada e saída (ao mesmo tempo).

## 8.6 Exercícios de Aprendizagem

1. Implemente um programa que leia as notas de $n$ alunos, onde $n$ é um valor digitado pelo usuário, e mostre quantas notas são maiores do que a média entre elas.

```
1  int main(){
2      int n, i, cont = 0;
3      cout << "Insira a quantidade de notas:\n";
4      cin >> n;
5      float notas[n], media = 0;
6      for(i = 0; i < n; i++){
7          cin >> notas[i];
8          media += notas[i];
9      }
10     media /= n;
11
12     for(i = 0; i < n; i++){
13         if(notas[i] > media) cont++;
14     }
15
16     cout << "acima da media: " << cont << endl;
17     return 0;
18 }
```

2. Implemente um programa que leia um número n do usuário e em seguida, armazene n notas em um vetor. O seu programa deve imprimir a posição da maior nota armazenada.

```
1  int main(){
2      int n, i, maiorpos;
3      cout << "Insira a quantidade de notas:\n";
4      cin >> n;
5      float notas[n], maior;
6      for(i = 0; i < n; i++){
7          cin >> notas[i];
8          if(i == 0){
9              maiorpos = i;
10         }
11         else{
12             if(notas[i] > notas[maiorpos]){
13                 maiorpos = i;
14             }
15         }
16     }
17     cout << "Pos. da maior nota: " << maiorpos << endl;
18
19     return 0;
20 }
```

3. Implemente duas funções:

   (a) Uma função chamada **leVet**, para ler os elementos de um vetor de números inteiros de tamanho n;

   (b) Uma função chamada de **imprimeVet**, para imprimir os elementos de um vetor de números inteiros em uma mesma linha da tela;

   Também, crie um programa (main) para utilizar as duas funções implementadas.

```
1  void leVet(int [], int );
2  void imprimeVet(int [], int );
3
4  int main()
5  {
6          int n;
7          cin >> n;
8          int v[n];
9          le_vet(v, n);
10         imprime_vet(v, n);
11         return 0;
12 }
13
14 void leVet(int vet[], int n)
15 {
16     for(int i = 0; i < n; i++){
17         cin >> vet[i];
18     }
19 }
20
21 void imprimeVet(int vet[], int n){
22     for(int i = 0; i < n; i++){
23         cout << vet[i] << ' ' ;
24     }
25 }
```

## 8.7 Exercícios de Fixação

1. Ler n elementos de uma matriz A tipo vetor e construir uma matriz B de mesma dimensão com os mesmo elementos da matriz A, sendo que deverão estar invertidos. Ou seja, o primeiro elemento de A passa a ser o último de B, o segundo elemento de A passa a ser o penúltimo elemento de B e assim por diante. Apresentar as matrizes A e B lado a lado.

2. Ler três matrizes (A, B e C) de uma dimensão com $n$ elementos cada. Construir uma matriz D, sendo esta a junção das três outras matrizes. Apresentar os elementos da matriz D.

3. Ler duas matrizes A e B de uma dimensão com $n$ elementos. A matriz A deverá aceitar apenas a entrada de valores pares, enquanto a matriz B deverá aceitar apenas a entrada de valores ímpares. A entrada das matrizes deverá ser validada pelo programa e não pelo usuário. Construir uma matriz C de forma que a matriz C seja a junção das matrizes A e B. Apresentar a matriz C.

4. Ler n elementos de vetor, colocá-los em ordem decrescente e apresentar os elementos ordenados.

5. Faça um programa que calcula a média das estaturas de uma turma com $n$ alunos. O program deve exibir um vetor com as estaturas abaixo da média e um outro vetor com as estaturas acima da média.

6. Implemente um programa em C++ que receba dois vetores de `float`, denominados `u` e `v`, cada um com `n` elementos. Assumindo $\mathbf{u} = [u_0, u_1, ..., u_{n-1}]^t$ e $\mathbf{v} = [v_0, v_1, ..., v_{n-1}]^t$

   (a) Implemete uma função para calcular o produto interno: o resultado é um número $p$, definido como:

   $$p = \mathbf{u}^t.\mathbf{v} = \sum_{i=0}^{n-1} u_i.v_i = u_0.v_0 + u_1.v_1 + ... + u_{n-1}.v_{n-1},$$

   (b) Implemete uma função para caclcular o produto elemento a elemento: o resultado é um vetor $y$, com o mesmo número de elementos de $u$ e $v$, onde:

   $$\mathbf{y} = [u_0 * v_0, u_1 * v_1, ..., u_{n-1} * v_{n-1}]^t$$

   Exemplo de execução:

```
Digite o numero de elementos dos vetores 'u' e 'v': 5
Digite os elementos do vetor 'u':
-2.1 0.4 -3.2 -1.2 4.8
Digite os elementos do vetor 'v':
0.7 -3.5 0.5 1.9 -2.2
Produto interno entre 'u' e 'v': -17.31
Produto elemento-a-elemento entre 'u' e 'v':
-1.47 -1.4 -1.6 -2.28 -10.56
```

7. A regressão linear por mínimos quadrados é um procedimento no qual os coeficientes $a_1$ e $a_0$ da equação $f(x) = a_1x + a_0$ são determinados de tal forma que essa função leve ao melhor ajuste de um determinado conjunto de pontos. Os coeficientes $a_1$ e $a_0$ são calculados utilizando as sequintes equações:

$$a_0 = \frac{\left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i\right) - \left(\sum_{i=1}^{n} x_i y_i\right)\left(\sum_{i=1}^{n} x_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$a_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

   Implemente uma função que receba como parâmetros de entrada os vetores $x$ e $y$, do conjunto de pontos, e como parâmetros de saída os coeficientes $a_1$ e $a_0$.

   Exemplo de execução:

```
x = [62 116 55 30 46];
y = [225 360 215 150 180];

a1 = 2.4691387  a0 = 73.407228
```

8. Implemente uma função que receba como parâmetro de entrada um vetor de números reais. A função a ser implementada deve retornar a quantidade de valores que são menores do que a média entre eles. A função `main` deve ler o tamanho `n` do vetor, cada um dos seus elementos e exibir na tela uma mensagem informando quantos elementos do vetor são maiores do que a média utilizando a função implementada.
**Exemplo de execução:**

```
Informe o tamanho do vetor:
5
Informe os elementos do vetor:
2.5 3.5 5.0 9.0 1.0
Elementos maiores do que a media: 2
```

9. Implemente uma função que receba como parâmetro de entrada um vetor de inteiros e como parâmetros de saída outros dois vetores de inteiros. A função a ser implementada deve armazenar no primeiro vetor de saída todos os números pares e no segundo vetor de saída todos os números ímpares. Observe que esta função também deve computar o tamanho dos vetores de saída, ou seja, a quantidade de números pares e ímpares: para isto, faça com que o tamanho de cada vetor de saída seja também um parâmetro de saída. A função `main` deve ler o tamanho do vetor, cada um dos seus elementos e, utilizando uma chamada à função implementada, exibir os vetores resultantes.
**Exemplo de execução:**

```
Informe o tamanho do vetor:
7
Informe os elementos do vetor:
0 1 2 3 4 5 6
Elementos pares:
0 2 4 6
Elementos impares:
1 3 5
```

10. Implemente uma função que recebe como parâmetros de entrada um vetor de caracteres e um caractere `c` de busca. A função deve retornar a posição da primeira ocorrência do caractere `c` no vetor ou -1 caso o caractere não exista no vetor. A função `main` deve ler o tamanho `n` do vetor, cada um dos seus elementos e exibir na tela uma mensagem informando a posição da primeira ocorrência do caractere especificado pelo usuário.
**Exemplo de execução:**

```
Informe o tamanho do vetor:
7
Informe os elementos do vetor:
a 2 t 4 e t $
Informe o caractere a ser buscado:
t
Posicao da primeira ocorrência de t: 2
```

## 8.8 Referências Bibliográficas


1. MANZANO, J.A.; OLIVEIRA, J.F.; **Algoritmos - Lógica para Desenvolvimento de Programação**. Editora Erica.

2. ASCENCIO, A.F.G.; CAMPOS, E.A.V. **Fundamentos da Programação de Computadores - Algoritmos, Pascal e C/C++**. 3ed. Editora Pearson.

3. DEITEL, H. M.; DEITEL, P. J. **C++ Como Programar**. 3ed. Editora Bookman.