

Profa. Michele Fúlvia Angelo

**PGCA 028 – Tópicos Especiais em Tecnologia
Computacional I – Introdução à Programação de
Computadores
Aula 5**

Universidade Estadual de Feira de Santana

Sumário

- Estruturas Homogêneas
 - Vetores
 - Ordenação de Vetores
 - Algoritmos de Busca
- Resolução de Exercícios

Novas Necessidades!

- Se o usuário precisar digitar uma sequência de números e depois mostrar esta sequência em ordem inversa. Como fazer?
- Para uma sequência de quatro números, quatro variáveis `numero1`, `numero2`, `numero3`, `numero4`
- E para uma sequência de 100 números? Que tal isso?

```
for(i=1; i<=N; i++){  
    scanf("%d",&numero); }
```

- Como armazenar vários valores em uma única variável?

Estruturas Homogêneas

- Estruturas homogêneas unidimensionais:
VETORES
- Estruturas homogêneas multidimensionais:
MATRIZES

Vetores

- Armazenam vários elementos do mesmo tipo primitivo;
- Vários valores em posições diferentes, com ordem específica entre si;
- Elementos podem ser repetidos ou não;
- Podemos encontrar esse tópico na literatura com as seguintes nomenclaturas: Vetor, Array, Matriz unidimensional ou Arranjo.

Declaração de Vetores

- Declaração:
tipo nomevariável[tamanho];

onde:

tipo: *qualquer tipo primitivo na linguagem C*

nomevariável: *nome do vetor*

tamanho: *define quantos elementos o vetor pode armazenar*

- Exemplos:

```
int lista[10];
```

```
float numeros[100];
```

Declaração de Vetores

- Declaração não-dimensionada:

Caso o tamanho não seja especificado, o vetor já deve ser inicializado.

```
int vet1[] = {1,5,8,3};
```

- O compilador alocará o espaço suficiente para armazenar o conteúdo atribuído, ou seja, o vetor terá tamanho = 4. Novos elementos podem ser inseridos depois.
- As duas notações a seguir são equivalentes:

```
int vet1[4] = {1, 2, 3, 4};  
int vet1[] = {1, 2, 3, 4};
```

Ilustração de um Vetor

- Um vetor capaz de armazenar 7 números inteiros, chamado **numeros**: `int numeros[7];`

numeros

0	
1	
2	
3	
4	
5	
6	

- ✓ Todas as posições pertencem ao vetor **numeros**, e cada uma dessas posições é capaz de armazenar um número inteiro;
- ✓ O acesso a cada um dos valores armazenados em **numeros** é feito por um índice que cada posição possui;
- ✓ Na linguagem C, esses índices iniciam-se no valor zero, ou seja, o índice da primeira posição de **numeros** é o zero e o índice da última posição de **numeros** é o 6.

Manipulação de Vetores

- Para ter acesso a cada elemento do vetor, é necessário especificar o nome do vetor seguido do índice da posição desejada, este último entre colchetes.
- Exemplo:
 - Armazenando o valor 15 na 4ª posição do vetor **numeros**:
`numeros[3] = 15;`
 - Lendo um número inteiro do teclado e armazenando-o na 1ª posição do vetor **numeros**:
`scanf("%d", &numeros[0]);`

Manipulação de Vetores

- Mais exemplos:

```
scanf("%d",&lista[0]);
```

```
printf("%d",lista[5]);
```

```
printf("%d",lista[x]);
```

```
lista[5]=lista[x+2];
```

- Não é possível fazer: **scanf("%d",&lista);**

Manipulação de Vetores

- `int lista[10];`

lista										
posição	0	1	2	3	4	5	6	7	8	9
valor										

```
lista[0]=50;  
lista[1]=100;  
lista[7]=-4;  
lista[2]=0;
```

lista										
posição	0	1	2	3	4	5	6	7	8	9
valor	50	100	0					-4		

Manipulação de Vetores

- Varrendo todos elementos:

```
int lista[10];
```

```
for(i=0; i<10; i++){  
    scanf("%d",&lista[i]);  
}
```

```
for(i=0; i<10; i++){  
    printf("%d",lista[i]);  
}
```

```
for(i=0; i<10; i++){  
    lista[i]=2*i;  
}
```

Considerações

- Tamanho do vetor é fixo (alocação estática), não é possível alterar.
- É preciso saber a quantidade máxima de elementos.
- Podemos manter uma variável numérica para sabermos quantas posições estão sendo usadas de fato.
- Nenhuma restrição é feita quanto a quantidade de valores inseridos. Se o usuário ultrapassar o limite, o programa tentará ler normalmente, mas o programa os escreverá em uma parte não alocada de memória. Isto pode resultar nos mais variados erros em tempo de execução.
 - Programa iria acessar local inválido da memória.
 - Erro (operação ilegal, segmentation fault) pode levar o programa a ser “finalizado” pelo sistema operacional.

Exemplos

- Programa que lê 10 valores reais e armazena-os em um vetor, e em seguida imprime os valores lidos.

```
#include <stdio.h>
main() {
    float exemplo[10];
    int i; //índice do vetor, sempre inteiro
    for(i = 0; i < 10; i++)
    {
        printf ("\nDigite o %d.o elemento do vetor: ", i + 1);
        scanf("%f",&exemplo[i]);
    }
    printf ("\nValores lidos:\n");
    for(i = 0; i < 10; i++)
        printf("%f\n", exemplo[i]);
}
```

Exemplos

- Programa que lê 7 valores inteiros e armazena-os em um vetor, e em seguida imprime os valores lidos em ordem inversa à leitura.

```
#include <stdio.h>
main()
{
    int exemplo[7],i;
    for(i=0; i < 7; i++)
    {
        printf("\nDigite o elemento %d: ", i + 1);
        scanf("%d", &exemplo[i]);
    }
    for(i=6; i>=0; i--)
        printf("\nElemento %d: %f", i+1, exemplo[i]);
}
```

Exemplos

- Não utilizando todas as posições do vetor:

```
#define MAX 100;
int main(){
    int i,n;
    int v[MAX];

    scanf("%d",&n); /* 0 <= n <= MAX */
    for(i=0; i<n; i++){
        scanf("%d",&v[i]);
    }
    for(i=n-1; i>=0; i--){
        printf("%d",v[i]);
    }
}
```


Exercícios

- 1) Programa que lê 8 valores inteiros e armazena-os em um vetor, e em seguida imprime a soma dos valores pares lidos.
- 2) Escreva um programa que leia 10 valores inteiros que representam notas e armazene-os em um vetor A. Em seguida calcule e imprima a média dos elementos do vetor e as notas que estão acima da média.
- 3) Escreva um programa que leia 30 números inteiros e armazene-os em um vetor A e leia também um inteiro n. Em seguida seu programa deve procurar o valor n em A e imprimir a posição em que este aparece. Se o elemento não estiver no vetor, o programa deve imprimir uma mensagem indicando que o elemento não pertence ao vetor.
- 4) Desenvolver um algoritmo que efetue a leitura de 10 elementos de um vetor A. Construir um vetor B, observando a seguinte lei de formação: se o valor do índice for par, o valor deverá ser multiplicado por 5, sendo ímpar, deverá ser somado com 5. Ao final, mostrar o conteúdo de B.

Algoritmos de Ordenação

- Algoritmo que coloca os elementos de uma dada sequência em uma certa ordem.
- Há três classes de algoritmos de ordenação interna:
 - Algoritmo de ordenação por troca
 - Algoritmo de ordenação por seleção
 - Algoritmo de ordenação por inserção

Algoritmo de Ordenação por Troca

- O método de ordenação baseado em troca, consiste em intercalar pares de itens que não estão em ordem até que não exista mais pares.
- O algoritmo ***bubble sort*** ou ordenação por flutuação (literalmente "por bolha") é um exemplo de ordenação por troca.
 - O princípio deste algoritmo é a troca de valores entre posições consecutivas, fazendo com que os valores mais altos (ou mais baixos) “borbulhem” para o final do vetor;
 - de simples entendimento e de fácil implementação;
 - está entre os mais conhecidos e difundidos métodos de ordenação de vetores;
 - não é um algoritmo eficiente.

Algoritmo *Bubble Sort*

```
int vet[] = {5,6,3,2,4};  
int i, j, aux;  
  
for (i=0; i<=3; i++)  
{  
    for (j=4; j=i+1; j--)  
    {  
        if (vet[j] < vet[j-1])  
        {  
            aux    = vet[j];  
            vet[j]  = vet[j-1];  
            vet[j-1] = aux;  
        }  
    }  
}
```

Algoritmo de Ordenação por Seleção

- O método de ordenação por seleção é levemente mais eficiente que o método da bolha.
- É um algoritmo que serve apenas para a ordenação de pequenos vetores.
- É um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os $(n-1)$ elementos restantes, até os últimos dois elementos.

Algoritmo Selection Sort

```
int vet[] = {5,6,3,2,4};  
int i, j, aux;
```

```
for (i=0; i<=3; i++)  
{  
    for (j=i+1; j<=4; j++)  
    {  
        if (vet[j] < vet[i])  
        {  
            aux    = vet[i];  
            vet[i] = vet[j];  
            vet[j] = aux;  
        }  
    }  
}
```

Algoritmo de Ordenação por Inserção

- O método de ordenação por inserção é o mais rápido entre os outros métodos considerados básicos (Bolha e Seleção).
- A principal característica deste método consiste em ordenar o vetor utilizando um sub-vetor ordenado localizado em seu início, e a cada novo passo, acrescentamos a este sub-vetor mais um elemento, até que atingimos o último elemento do vetor, fazendo assim com que ele se torne ordenado.

Algoritmo de Ordenação por Inserção

```
int vet[] = {5,6,3,2,4};  
int i, j, aux;
```

```
for (i=1; i<=4; i++)  
{  
    aux = vet[i];  
    j = i;  
    while ((j > 0) && (vet[j-1] > aux))  
    {  
        vet[j] = vet[j-1];  
        j = j - 1;  
        vet[j] = aux;  
    }  
}
```


Algoritmos de Busca

- Dada uma coleção de n elementos, pretende-se saber se um determinado valor está presente nessa coleção.
- A busca do valor pode ocorrer de formas diferentes, dentre elas:
 - Busca Sequencial
 - Busca Binária

Busca Sequencial

- Uma solução possível é percorrer o vetor desde a primeira posição até a última. Para cada posição i , comparamos `vetor[i]` com `valor`.
 - Se forem iguais, dizemos que `valor` existe.
 - Se chegarmos ao fim do vetor sem sucesso, dizemos que `valor` não existe.

Algoritmo para Busca Sequencial

```
int i, posicao=-1, valorProcurado=3, tam = 5;  
int v[] = {5,6,3,2,4};  
  
for (i = 0; i < tam; i++) {  
    if (v[i] == valorProcurado) {  
        posicao = i;  
    }  
}  
if (posicao==-1)  
    printf("Valor não encontrado!");  
else  
    printf("Valor encontrado na posicao %d!", posicao);
```

Busca Binária

- Aplicável a uma lista **ordenada**, para que seja possível a comparação: $A[i] < A[j]$?
- Portanto, comparando um determinado elemento com o elemento procurado, saberemos:
 - se o elemento procurado é o elemento comparado,
 - se ele está antes do elemento comparado ou
 - se está depois.

Busca Binária

- Se fizermos isso sempre com o elemento do meio da lista, a cada comparação dividiremos a lista em duas, reduzindo nosso tempo de pesquisa.
- Se em um determinado momento o vetor, após sucessivas divisões, tiver tamanho zero, então o elemento não está no vetor.

Algoritmo para Busca Binária

```
int valorProcurado=6, tam = 5, inf=0, sup=tam-1, meio, posicao=-1;  
int v[] = {1,4,6,8,9};
```

```
while (inf <= sup)  
{  
    meio = inf + (sup-inf)/2;  
    if (valorProcurado == v[meio]){  
        posicao = meio;  
        inf = sup+1;  
    }  
    else  
        if (valorProcurado < v[meio])  
            sup = meio-1;  
        else  
            inf = meio+1;  
}  
if (posicao==-1)  
    printf("Valor não encontrado!");  
else  
    printf("Valor encontrado na posicao %d!", posicao);
```