

Vetores em C

Disciplina: PROGRAMAÇÃO II

Prof. Jean Eduardo Glazar
Curso de Sistemas de Informação
Campus Colatina-ES



INSTITUTO FEDERAL
Espírito Santo

Vetores (ou *Array*)

- Tipo de variável que permite armazenar diversos valores.
- Os valores DEVEM ser do mesmo tipo.
- Cada valor é identificado por um índice (posição).
- O índice começa de ZERO. **Exemplo:** um vetor com 10 elementos é identificado pelos índices de 0 a 9.
- DEVE-SE declarar com um tamanho máximo.

Declaração de Vetores

- ▶ A declaração de vetores é semelhante à declaração de variáveis, apenas adicionando o **tamanho** do vetor entre colchetes:

<tipo> **<nome_vetor>** [**<tamanho>**] ;

Onde:

<tamanho> ➔ quantidade de elementos do vetor.

Declaração de Vetores

```
int vetor[10];
```

- ▶ Os índices do vetor começam em zero.
- ▶ Logo, no exemplo acima os índices vão de 0 a 9.

vetor	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Vetores – Exemplo

```
int main () {  
    int vet1[2], vet2[2], soma[2];  
    printf ("Entre com 2 numeros: ");  
    scanf ("%d", &vet1[0] );  
    scanf ("%d", &vet1[1] );  
  
    printf (" Entre com mais 2 numeros : ");  
    scanf ("%d", &vet2[0] );  
    scanf ("%d", &vet2[1] );  
    soma[0] = vet1[0] + vet2[0];  
    soma[1] = vet1[1] + vet2[1];
```

```
    printf ("\n\n");  
    if ( soma[0] > soma[1] ) {  
        printf("A soma dos números iniciais  
        é maior que a soma dos finais");  
    } else {  
        printf("A soma dos números iniciais  
        não é maior que a soma dos  
        finais");  
    }  
}
```



Tamanho do *Array*

- ▶ Não existe uma função para saber quantos elementos foram armazenados.
- ▶ Por isso que devemos utilizar uma variável para armazenar essa quantidade.
- ▶ Veja que essa quantidade armazenada deve ser menor que o tamanho máximo na declaração do vetor.
- ▶ Então temos DUAS informações diferentes:
 - ▶ O tamanho MÁXIMO
 - ▶ A QUANTIDADE realmente armazenada

Tamanho do *Array*

```
int vetor[10];
```

```
int qtde = 0;
```

```
vetor[qtde] = aleatorio();
```

```
qtde++;
```

```
vetor[qtde] = aleatorio();
```

```
qtde++;
```

```
vetor[qtde] = aleatorio();
```

```
qtde++;
```

vetor

0	5
1	28
2	15
3	
4	
5	
6	
7	
8	
9	

Tamanho do *Array*

- ▶ Como o tamanho máximo não varia, podemos utilizar uma CONSTANTE.

```
#define MAX 100
```

```
int main() {  
    int vetor[MAX];  
    int qtde = 0;  
    vetor[qtde] = aleatorio();  
    qtde++;  
    . . .  
}
```


Vetores como ponteiros

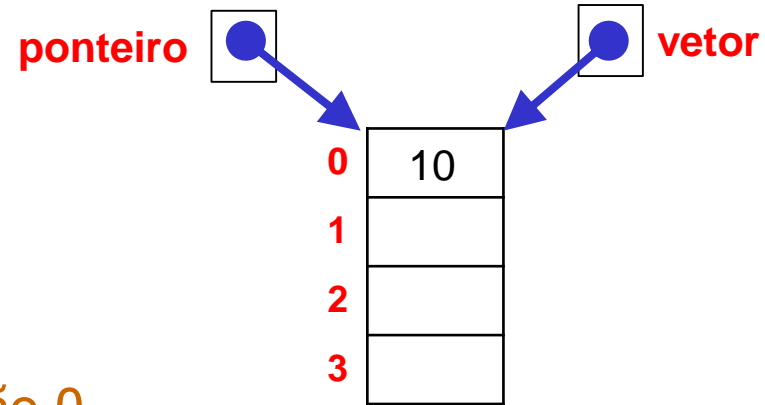
- ▶ Já vimos que um ponteiro é utilizado quando queremos simular passagem por referência em C, mas os ponteiros são muitos mais poderosos que isso.
- ▶ Todo vetor na realidade é um ponteiro para um conjunto de variáveis de um mesmo tipo onde cada variável é acessada por um único índice.

Todo vetor é um ponteiro para a primeira posição.



Vetores como ponteiros – Exemplo

```
int main() {  
    int vetor[4];  
    int *ponteiro;  
    ponteiro = vetor;  
    ponteiro[0] = 10;  
    printf("O conteúdo da posição 0  
           do vetor é: %d \n", vetor[0] );  
}
```



Vetores como parâmetros

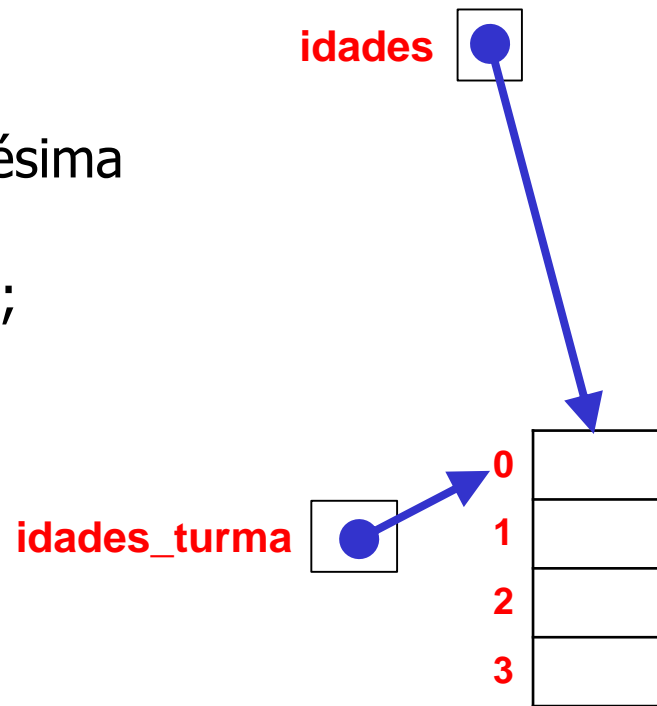
- ▶ Utiliza-se ponteiros para passar um vetor como parâmetro para função.

É fundamental que a função conheça o **tamanho** do vetor, caso contrário, ela pode invadir espaços de memória que não são do vetor, o que pode gerar travamentos de software.



Vetores como parâmetros – Exemplo

```
void lerDados(int *idades, int qtde) {  
    int i;  
    for (i = 0; i < qtde; i++) {  
        printf("Entre com a %d-ésima  
            idade\n", i);  
        scanf("%d", &idades[i] );  
    }  
}  
  
int main() {  
    int idades_turma[4];  
    lerDados(idades_turma, 4);  
}
```



Tamanho indefinido

- ▶ Quando não sei quantos elementos serão lidos, declarar-se o vetor com um tamanho grande.
- ▶ Na função que lê os dados, a quantidade deve ser passada como referência, para que possa ser adicionada.
- ▶ Cuidado para não ultrapassar o valor máximo do vetor.

Leitura de dados – Exemplo

```
void lerDados(int *vetor, int *qtde) {  
    char cont;  
    do {  
        vetor[*qtde] = lerIdade();  
        (*qtde)++;  
        printf("Deseja continuar? ");  
        scanf(" %c", &cont);  
    } while ( toupper(cont) == "S");  
}
```

```
#define MAX 100  
  
int main() {  
    int idadesTurma[MAX];  
    int qtde = 0;  
  
    lerDados(idadesTurma, &qtde);  
    . . .  
}
```

Imprimir

```
void imprimir (int *vetor, int qtde) {  
    int i;  
    for (i = 0; i < qtde; i++) {  
        printf("%d ", vetor[i]);  
    }  
}  
  
int main() {  
    int idadesTurma[MAX];  
    int qtde = 0;  
    lerDados(idadesTurma, &qtde);  
    imprimir(idades_turma, qtde);  
}
```

Pesquisar

```
int pesquisar (int *vetor, int qtde, int pesq) {  
    int i;  
    for (i = 0; i < qtde; i++) {  
        if (vetor[i] == pesq)  
            return i;  
    }  
    return -1;  
}
```

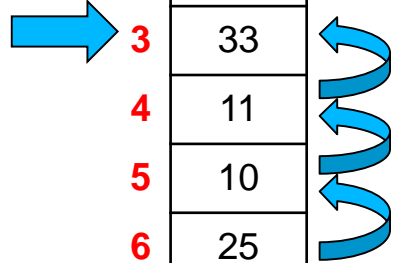

Remover

- ▶ Não existe uma função que retira um elemento do um vetor.
- ▶ O que devemos fazer é deslocar os elementos uma posição, para sobrescrever aquele que queremos retirar.
- ▶ E diminuir a quantidade total de elementos

Remover

antes **qtde** 7

0	5
1	28
2	15
3	33
4	11
5	10
6	25
7	
8	
9	



remover(3):

Deslocar 4 → 3

Deslocar 5 → 4

Deslocar 6 → 5

depois **qtde** 6

0	5
1	28
2	15
3	11
4	10
5	25
6	
7	
8	
9	

Remover

```
void remover (int *vetor, int *qtde, int pos) {  
    int i;  
    for (i = pos; i < *qtde- 1; i++) {  
        vetor[i] = vetor[i+1];  
    }  
    (*qtde)--;  
}
```

Alocação Dinâmica X Alocação Estática

- ▶ A **alocação estática** é utilizada quando é sabido o tamanho de um vetor, ou seja, quando conhece-se previamente a quantidade de espaço que será necessária para receber todos os dados.
- ▶ A **alocação estática** pode ser bastante ineficiente e restritiva no que tange o uso de memória, o exemplo seguinte ilustrará esse problema.



Alocação Estática – Exemplo

```
int main() {  
    int numeros[1000];  
    int quant_numeros, i;  
    printf("Entre com a quantidade de  
           números: ");  
    scanf("%d", &quant_numeros);  
    for (i = 0; i < quant_numeros; i++) {  
        printf("Entre com o %d-ésimo  
              número: ", i);  
        scanf("%d", &numeros[i] );  
    }  
}
```

Desperdício de

memória: suponha que quant_numeros seja 2. Foram alocadas 1000 posições em números e apenas as duas primeiras serão utilizadas.



Invasão de memória:

Suponhamos agora que quant_numeros seja 1001, não há como colocar 1001 números nas 1000 posições. Isso pode gerar travamentos de software

Alocação Dinâmica X Alocação Estática

- ▶ Para resolver esse problema existe a alocação dinâmica.
- ▶ Na **alocação dinâmica**, o tamanho do vetor é criado durante a execução do programa.
- ▶ Existem algumas funções para alocação dinâmica em C. Trabalharemos apenas a função **malloc**, por entender que ela atende ao que necessitaremos para a alocação dinâmica.



Função malloc

```
void *malloc (unsigned int <tamanho> )
```

Onde:

<tamanho> → é o número de bytes que se deseja alocar, ou seja, o tamanho do vetor **em bytes**.

Função malloc

- ▶ Definir o tamanho do vetor por bytes pode não ser uma tarefa das mais simples (dependendo do compilador, cada tipo primitivo pode ter um tamanho). Para resolver esse problema utiliza-se o operador **sizeof**, que é definido da seguinte forma:

int sizeof (<tipo que se deseja o tamanho>)

- ▶ **Exemplos:** sizeof(int) → retorna 4 bytes
sizeof(char) → retorna 1 byte

Função malloc

- Dessa forma é muito comum o uso do **malloc** com o **sizeof** para definir vetores dinâmicos, vejamos então uma forma relativamente padronizada para definir vetores dinâmicos:

vetor = (<tipo> *) malloc (<tamanho> * sizeof (<tipo>))

Quantidade de elementos

Tamanho em bytes de UM elemento

malloc – Exemplo

```
int main() {  
    int *numeros;  
    int quant_numeros, i;  
    printf("Entre com a quantidade de números: ");  
    scanf("%d", &quant_numeros);  
    numeros = (int *) malloc (quant_numeros * sizeof (int) );  
    for(i = 0; i < quant_numeros; i++) {  
        printf("Entre com o %d-ésimo número: ", i);  
        scanf("%d", &numeros[i] );  
    }  
}
```

Função calloc

- ▶ A função **calloc** é semelhante à **malloc**, a diferença é que o conteúdo da memória alocada é inicializada com ZERO. A sintaxe possui uma pequena diferença na passagem dos parâmetros:

vetor = (<tipo> *) calloc (<tamanho> , sizeof (<tipo>))

Quantidade de elementos

Tamanho em bytes de UM elemento

Função realloc

- ▶ A função **realloc** serve para realocar mais espaço de memória, mantendo o conteúdo anterior.
- ▶ Ela deve ser usada passando um ponteiro que já foi alocado anteriormente. Veja a sintaxe:

vetor = (**<tipo> ***) **realloc** (**vetor** , **<tamanho> * sizeof (<tipo>)**)

Quantidade de elementos anterior mais novo espaço

Tamanho em bytes de UM elemento



**INSTITUTO
FEDERAL**
Espírito Santo