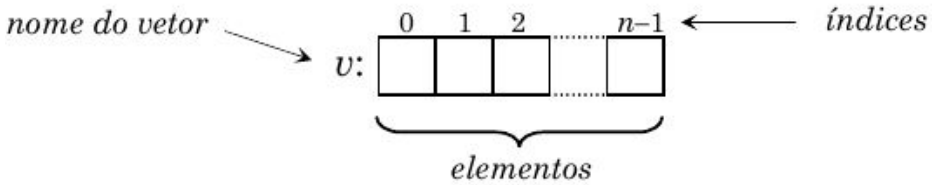
Linguagem de Programação

Arrays

Arrays

- Um Array (ou *vetor)* é uma coleção de variáveis de um mesmo tipo, que compartilham o mesmo nome e que ocupam posições consecutivas de memória.
- Cada uma dessas variáveis é identificada por um índice.
  - Se **v** é um vetor com *n* posições, seus elementos são **v[0]**, **v[1]**, **v[2]**, ..., **v[n−1]**.



Em C os vetores são sempre indexados a partir de zero e, portanto, o última elemento de um vetor de tamanho *n* ocupa a posição *n-1* do vetor.

Declarando vetores

- Um vetor para armazenar 5 números inteiros pode ser criado da seguinte maneira: `int v[5];`
- Um vetor pode ser indexado com qualquer expressão cujo valor seja inteiro
  - Por ex. considere i=5:

```
① w[0] = 17;
② w[i/2] = 9;
③ w[2*i-2] = 95;
④ w[i-1] = w[8]/2;
```

```
⑤ w[i] = w[2];
⑥ w[i+1] = w[i]+w[i-1];
⑦ w[w[2]-2] = 78;
⑧ w[w[i]-1] = w[1]*w[i];
```

- O C não verifica a consistência dos valores usados como índices.
  - Qualquer valor pode ser usado como índice, mesmo que seja inadequado
  - É responsabilidade do programador definir corretamente os índices

Inicializando um vetor

- Inicializando um vetor sem especificar a quantidade de elementos
  - *int valores[ ] = {3,5,7}*
- Iniciando apenas alguns elementos do vetor:
  - *int valores[5] = {2,4,6}* será equivalente a *int valores[5] = {2,4,6,0,0}*
  - posições não preenchidas recebem valor zero
- Operador **sizeof()**
  - retorna o tamanho em bytes que uma variável está utilizando na memória
    - Também retorna o tamanho de tipos
  - Ex.: int v[] = { 3, 4, 6, 7}
    - o número de elementos de v será sizeof(v)/sizeof(int)
    - Uma variável inteira equivale a 4 bytes, então, o nº de elementos em v será (4+4+4+4)/4

Vetores como argumento de funções

```c
#include <stdio.h>
void funcao(int v[]); //protótipo. Não
necessita explicitar o tamanho do vetor
int main (void) {
  int v[3];
  v[0]=12;
  v[1]=13;
  funcao(v);
  return 0;
}
void funcao(int v[3]){
     printf("v[1]=%d v[2]=%d\n", v[0],v[1]);
}
```

- Assim como nas outras variáveis, a cada chamada de função, é criada uma cópia do vetor passado como parâmetro
  - a função não manipula o vetor original, mas uma cópia sua

- Para manipular o vetor original, esse deve ser passado por **referência**
  - estudaremos isso no assunto sobre **ponteiros**

Exercício

Escreva um programa em C para contar o número total de elementos duplicados em um array.

- ***Dados de teste :***
  Insira o número de elementos a serem armazenados no array: 3
  Insira 3 elementos no array:
  elemento - 0: 5
  elemento - 1 : 1
  elemento - 2 : 1

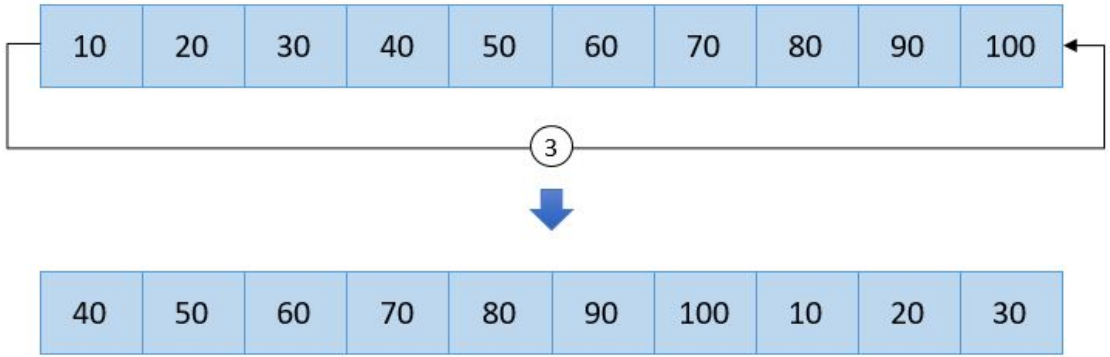
- ***Saída esperada:***
  O número total de elementos duplicados encontrados no array é: 1

Exercício

- Escreva um programa que ordene um vetor (inteiros) de maneira não-decrescente, utilizando o *método bolha.*

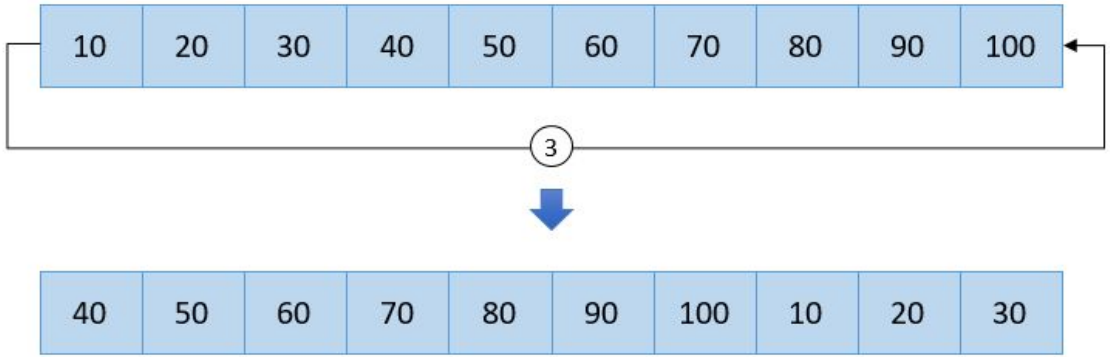
Exercício

- Escreva um programa em C para rotacionar à esquerda um vetor em *n* posições.
  - Ex: n = 3
  - Array inicial: 1 2 3 4 5 6 7 8 9 10
  - Array rotacionado: 4 5 6 7 8 9 10 1 2 3



Exercício

- Escreva um programa em C para rotacionar à esquerda um vetor em *n* posições.
  - Ex: n = 3



- **Ideia:** copiar todo elemento[i+1] para elemento[i], e por fim copiar o 1º elemento para a última posição

Matrizes

Matrizes

- Uma matriz é uma coleção homogênea, geralmente bidimensional, cujos elementos são distribuídos em linhas e colunas
- Se **A** é uma matriz *m* x *n*, então suas linhas são indexadas de 0 a *m*-1 e suas colunas de 0 a *n*-1
- Para acessar um um elemento particular de **A**, fazemos **A[*i*][*j*]**
  - *i*: linha e *j*: coluna

- Declarando uma matriz 3x4 de inteiros: `int A[3][4];`
  - Essa declaração cria um vetor **A**, cujos elemento **A[0]**, **A[1]** e **A[2]** são também vetores. Cada um deles, contendo 4 elementos do tipo *int*.

Matriz

- Para processarmos uma matriz, fazemos uso de **for**s aninhados:

  - Armazenando valores em uma matriz:

```
int A[3][4];
int i, j;
for(i=0; i<3; i++) {
   for(j=0;j<4;j++) {
     printf(" ler A[%d][%d]:", i, j);
      scanf("%d", &A[i][j]);
     }
  }
```

  - Mostrando a matriz

```
for(i=0; i<3; i++) {
 for(j=0;j<4;j++)
    printf("%d ", A[i][j]);
 printf("\n");
}
```

Inicialização de matrizes 2D

```
int A[2][4] = {
   {10, 11, 12, 13},
   {14, 15, 16, 17}
};
```

OU

```
int A[2][4] = {10, 11,
12, 13, 14, 15, 16, 17};
```

- Ambas as declarações são válidas. **PORÉM**, recomenda-se utilizar a 1ª, que é mais legível (melhor visualização das linhas e colunas)

Inicialização de matrizes 2D

- Em vetores, não é necessária a especificação do tamanho na declaração
- Em matrizes 2D, essa especificação é **sempre necessária** para a **2ª dimensão**
  - Exemplos:

```c
/* declaração válida */
int abc[2][2] = {1, 2, 3 ,4 };

/* declaração válida */
int abc[][2] = {1, 2, 3 ,4 };

/*Declaração inválida - você deve especificar a 2ª dimensão*/
int abc[][] = {1, 2, 3 ,4 };

/* Inválida pela mesmo motivo mencionado acima*/
int abc[2][] = {1, 2, 3 ,4 };
```

Exercício

- Crie um programa em C que preencha uma matriz 5x5 com valores aleatórios (0 a 9) e a mostre na tela. Em seguida, encontre a sua transposta.
    - Obs.: matriz transposta é a matriz que se obtém da troca de linhas por colunas
    - Ex.:

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

Exercício

Crie um programa em C para encontrar a multiplicação dos elementos da diagonal principal de uma matriz. Essa matriz deve ser 5x5 e preenchida com valores aleatórios.