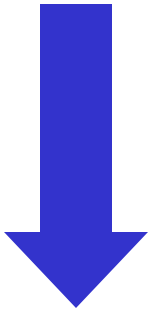
Programa = Algoritmo + Estrutura de Dados



A todo momento durante a execução de qualquer tipo de programa os computadores estão manipulando informações, representadas pelos diferentes tipos de dados, armazenadas em sua memória.

Tipos de dados

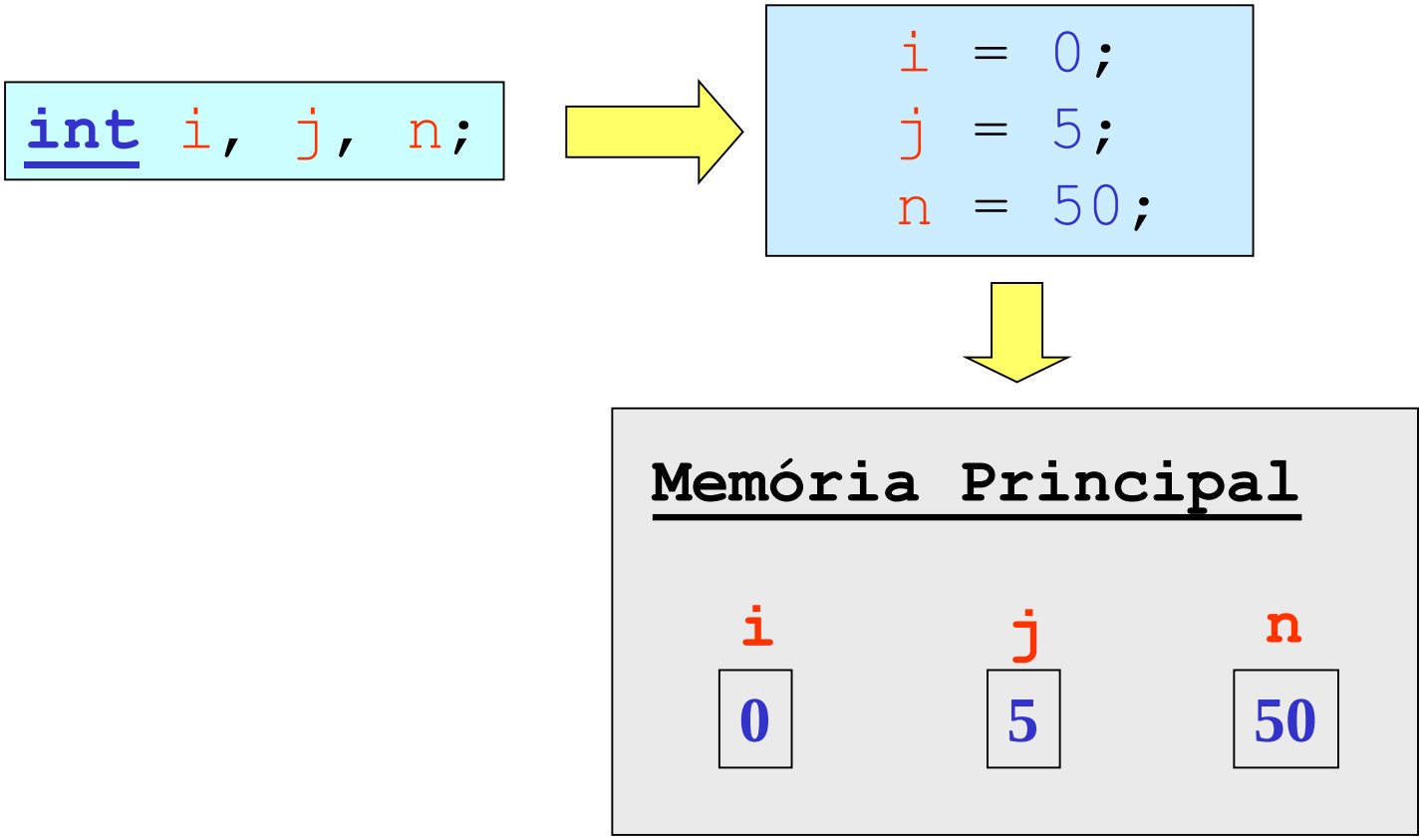
- valores que uma variável pode assumir
  - representação interna desses valores
- operações que podem ser realizadas com essa variável

Classificação dos tipos de dados

- Tipos de Dados Simples
  - Tipos de Dados Inteiros: *int*, *long* e *unsigned int*
  - Tipos de Dados Reais: *float* e *double*
  - Tipos de Dados Caracteres: *char*
- Tipos de Dados Estruturados
  - cadeia de caracteres *(string)*, vetores *(array)*, registros *(struct)* e ficheiros em disco.
- ponteiros (alocação dinâmica de memória)
- Classes e Objetos (Orientação a Objetos)

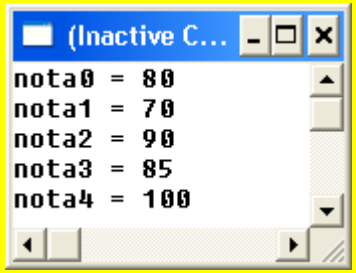
Tipos de Dados Simples

Uma variável simples é uma entidade criada para permitir o acesso a uma posição de memória onde se armazena uma informação de um determinado tipo de dado pela simples referência a um nome simbólico.



Utilizando variáveis de Tipos de Dados Simples:

```c
#include "stdio.h"

void main() {
   int nota0 = 80;
   int nota1 = 70;
   int nota2 = 90;
   int nota3 = 85;
   int nota4 = 100;

   printf("nota0 = %d\n", nota0);
   printf("nota1 = %d\n", nota1);
   printf("nota2 = %d\n", nota2);
   printf("nota3 = %d\n", nota3);
   printf("nota4 = %d\n", nota4);
}
```



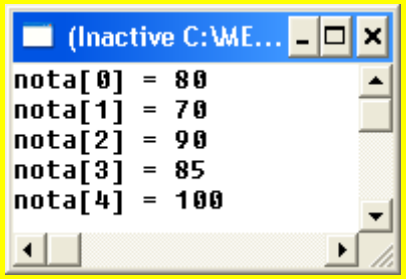
Tipos de Dados Estruturados

- armazenam **diversos** itens de uma só vez

- isto significa:
  - em uma mesma estrutura de dados, é possível ter diversas variáveis de tipos de dados simples agrupadas

Utilizando variáveis de Tipos de Dados Estruturados:

```c
#include "stdio.h"

void main() {

// declarando e alimentando o vetor
   int nota[5] = {80, 70, 90, 85, 100};

// percorrendo todos os valores
// armazenados no vetor
   for ( i=0; i<5; i++)
{
     printf("nota[%d] = %d\n", i, nota[i]);
}
```



Vetores 1/3

- representa um conjunto de valores do mesmo tipo (**estrutura homogênea**), referenciáveis pelo mesmo nome e individualizados entre si através de sua posição dentro desse conjunto (**variáveis indexadas**)
- uma sequência de endereçamentos (posicoes contíguas) de memória para armazenar os conteúdos de suas diversas posições dentro do vetor
- sinônimos: variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, matrizes (unidimensionais), tabelas em memória ou **arrays**.

Vetores 2/3

Vetores são estruturas de dados que armazenam usualmente uma quantidade fixa de dados de um certo tipo; por esta razão, também são conhecidos como estruturas homogêneas de dados.

Internamente, um vetor armazena diversos valores, cada um associado a um número que se refere à posição de armazenamento, e é conhecido como índice. Os vetores são estruturas indexadas, em que cada valor que pode ser armazenado em uma certa posição (índice) é chamado de elemento do vetor.

Cada elemento do vetor pode ser utilizado individualmente de forma direta, ou seja, pode ser lido ou escrito diretamente, sem nenhuma regra ou ordem preestabelecida, fazendo dos vetores estruturas de dados de acesso aleatório.

Vetores 3/3

O número de posições de um vetor corresponde ao tamanho que ele tem; assim, um vetor de tamanho 10 tem esse número de elementos, isto é, pode armazenar até dez elementos distintos. Os diferentes elementos de um vetor são distinguidos unicamente pela posição que ocupam no vetor. Cada posição de um vetor é unicamente identificada por um valor inteiro positivo, linear e sequencialmente numerado. Ou seja:

vetor[i] "i-ésimo" elemento do vetor, sendo que o valor da variável "i" deve pertencer ao intervalo do índice do vetor, ou seja, $0 \leq i \leq (n-1)$

As linguagens C e C++ como o Java são linguagens com vetores *zero-based*, isto é, as posições do vetor iniciam a numeração a partir do valor "0", portanto, um vetor de tamanho 10 teria índices iniciados em 0 prosseguindo até o 9.

Vetores ou Variáveis Indexadas

- O termo *indexada* provém da maneira como é feita a individualização dos elementos do conjunto de dados: por meio de **índices**.
- Uma variável indexada pode ser definida como tendo um ou mais índices.
  - **vetor**, um único índice (unidimensionais)
  - **matriz**, dois índices (bidimensionais)
  - multidimensionais, pouco freqüentes

Em Síntese: Características Básicas

- estrutura de dados homogénea e indexada
- todos os elementos da estrutura são igualmente acessíveis
  - tempo e tipo de procedimento para acessar qualquer um dos elementos do vetor são “iguais”
- cada elemento componente deste tipo de estrutura de dados tem um nome próprio que é o nome do vetor seguido da posição, ou índice

  NomeDaVariávelVetor[posição, ou índice]

Declarando Variáveis do Tipo Vetor

Na declaração de vetores deverão ser fornecidas três informações: **(1)** o nome do vetor, **(2)** o número de posições do vetor (seu tamanho) e **(3)** o tipo de dado que será armazenado no vetor.

A declaração de um vetor para “inteiros”, de nome “vetor” e tamanho igual a 10, em C:

```
#define n 10;          // tamanho do vetor
 int vetor[n];         // declaração do vetor
```

Podemos notar que as declarações de vetores são semelhantes às declarações de variáveis, os elementos sintáticos que diferenciam as variáveis do tipo vetor das outras variáveis são os parentheses retos.

Declaração de vetores ou variáveis indexadas:

```
#define n 100;
float salarios[n];

int a[10], b[10]; // declaração compacta
```

Nos exemplos acima temos:

a e **b**, são conjuntos de **10** componentes inteiros cada, ou seja, variáveis capazes de armazenar **10** números inteiros simultaneamente;

salarios, com capacidade de armazenar um total de **n** (**100**) números reais;

vetores, individualizando valores:

x [expressão]

onde:

x nome da variável do tipo vetor

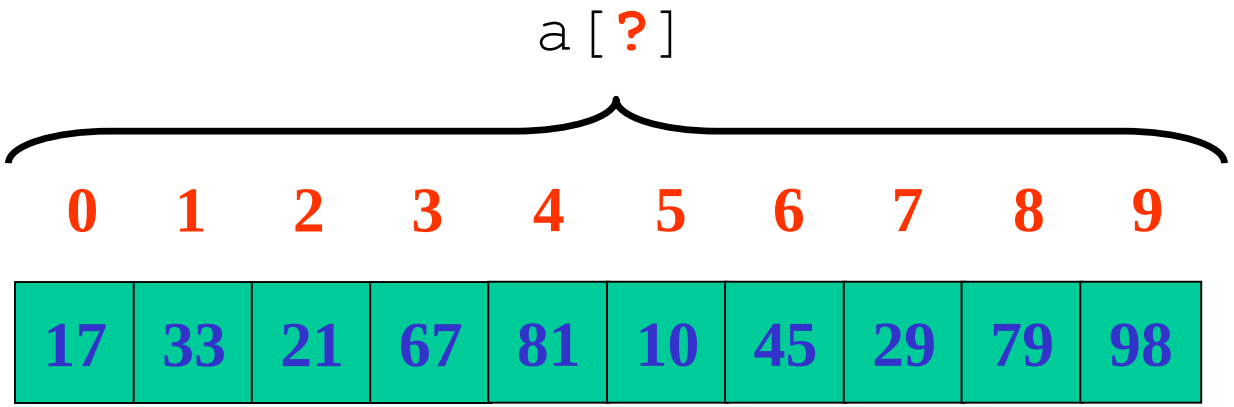
expressão **posição** que define qual o elemento da estrutura de dados está sendo referenciado. ***Atenção:*** *deve ser um valor pertencente ao intervalo do índice da variável.*

Observação: Os elementos de um vetor tem todas as características de uma variável comum e podem aparecer livremente em expressões e atribuições.

```
vetor[10] = vetor[1] * vetor[i + j];
```

Compreendendo o armazenamento de um vetor:

```
void main() {

#define n  10;
   int a[n];

   a[0] = 17;
   a[1] = 33;
   a[2] = 21;
   a[3] = 67;
   a[4] = 81;
   a[5] = 10;
   a[6] = 45;
   a[7] = 29;
   a[8] = 79;
   a[9] = 98;
}
```



Operações básicas com **vetores** ou **array's**:

Do mesmo modo que acontece com variáveis simples, também é possível realizar operações de atribuição (**=**), leitura (**scanf**) e escrita (**printf**) com variáveis indexadas. Contudo, não é possível operar diretamente com o conjunto completo, mas com cada um de seus elementos isoladamente.

Por exemplo, para somar dois vetores é necessário somar cada um de seus componentes dois a dois. Da mesma forma, as operações de atribuição, leitura e escrita de conjuntos devem ser feitas elemento a elemento.

sendo, `int a[10], b[10], c[10];`

Certo:

```
for (int i=0; i<10; i++)
  c[i] = a[i] + b[i];

for (int i=0; i<10; i++)
  scanf("%d", &a[i]);

for (int i=0; i<10; i++)
  printf("%d\n", a[i]);
```

Errado:

```
c = a + b;

scanf("%d", &a);

printf("%d", a);
```

Atribuição (=)

No caso de variáveis indexadas, além do nome da variável deve-se necessariamente fornecer também o(s) índice(s) que individualizam o elemento do conjunto onde será armazenado o resultado da avaliação da expressão.

NomeDaVariávelVetor[posição] = Expressão;

```
void main() {
   int nota[5];
   nota[0] = 80;
   nota[1] = 70;
   nota[2] = 90;
   nota[3] = 85;
   nota[4] = 100;
   ...
}
```

Leitura

A leitura de um conjunto é feita passo a passo, um componente por vez, usando a mesma sintaxe da instrução primitiva de entrada de dados (**scanf**). Mais uma vez, além do nome do conjunto, deve ser explicitada a posição do elemento no vetor que receberá o valor lido.

```
#include "stdio.h"
void main() {
   int nota[5];

   for (int i=0; i<5; i++) {
      printf("\nInforme o valor da %da. Nota:", i);
      scanf("%d", &nota[i]);
   }
   ... // processo de saída ou escrita
}
```

Escrita

A escrita de um conjunto obedece à mesma sintaxe da instrução primitiva de saída de dados (**printf**). Mais uma vez, convém lembrar que, além do nome do conjunto, deve-se também especificar por meio de seu(s) índice(s) qual o elemento do vetor será escrito.

```
#include "stdio.h"

void main() {
  int nota[5];

  ... // processo de entrada ou leitura

  for (int i=0; i<5; i++)
    printf("\n%d- %d", i, nota[i]);
 }
```

Vetores Bidimensionais, ou Matrizes

C suporta vetores multidimensionais. A forma mais simples de vetor multidimensional é o vetor bidimensional, ou matriz ($A_{i,j}$).

– tabela de linhas e colunas

– armazenam vetores dentro de vetores

Por exemplo, para declarar um vetor bidimensional de inteiros denominado "matriz" de tamanho **2** x **4**, ou seja, **2** linhas por **4** colunas, deve-se escrever o seguinte código:

```
int matriz[2][4];
```

Atenção: Muitas linguagens de programação usam vírgulas para separar as dimensões do vetor multidimensional; C, em contraste, coloca cada dimensão no seu próprio conjunto de parenteses retos.

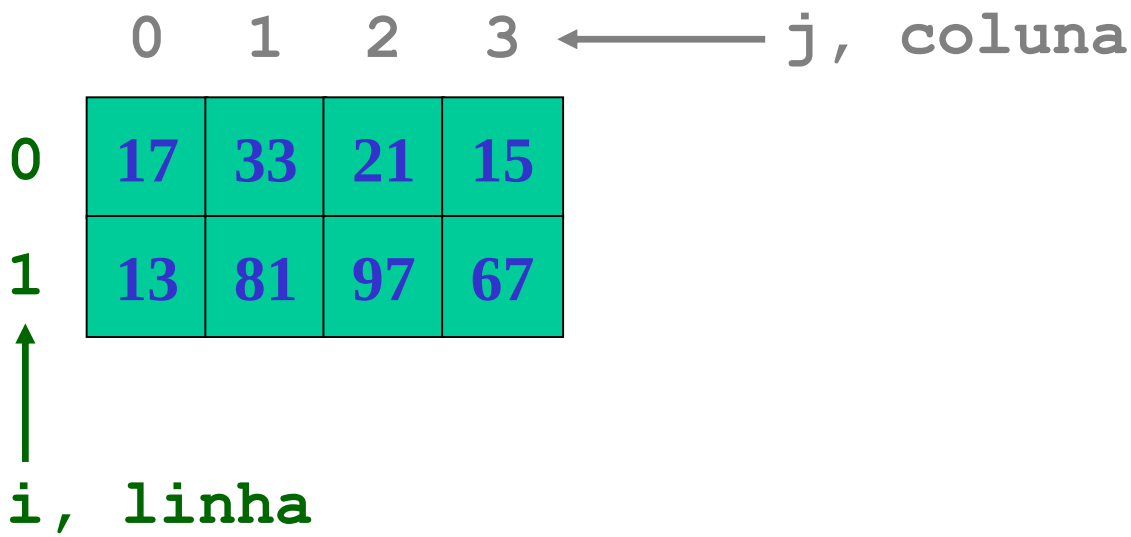
Declarando um vetor bidimensional, ou matriz

```
int A[2][4];
```

Referenciando as posições da matriz

1a. Linha	2a. Linha
A[0][0] = 17;	A[1][0] = 13;
A[0][1] = 33;	A[1][1] = 81;
A[0][2] = 21;	A[1][2] = 97;
A[0][3] = 15;	A[1][3] = 67;

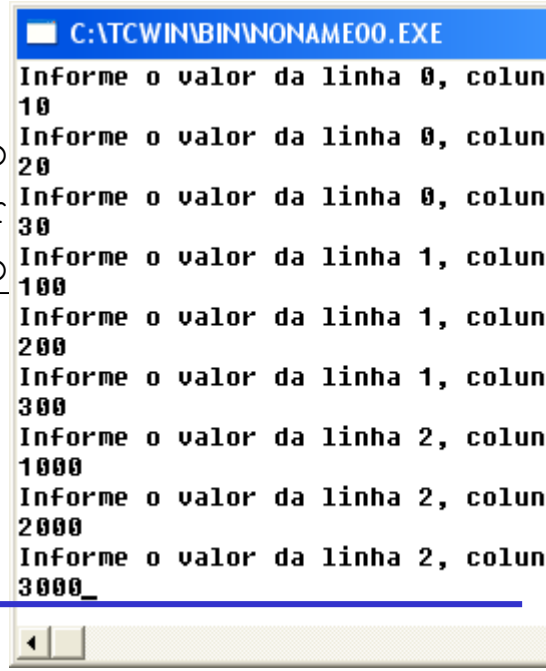
A[i][j]

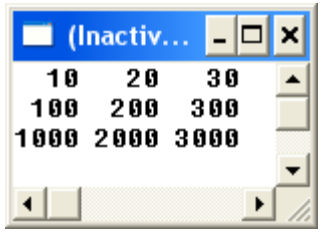


Ler e escrer um vetor bidimensional, ou matriz

A leitura de um conjunto bidimensional é feita, unidimensionais, passo a passo, um componente p mesma sintaxe das instruções primitivas de entr saída (**printf**) informando o nome da matriz as p e da coluna como mostra o exemplo a seguir:

```
#include "stdio.h"
#include "conio.h"

void main() {
  const n = 3;
  int matriz[n][n], i, j;
  for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
      printf("Informe o valor da linha %d, coluna %d.\n", i, j);
      scanf("%d", &matriz[i][j]);
    }
  system("cls");
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
      printf("%4d ", matriz[i][j]);
    printf("\n");
  }
}
```





Inicialização de Vetores

C permite a inicialização de vetores no momento da declaração, como é demonstrado nos exemplos a seguir:

```
int vetor[10] = {17, 33, 21, 67, 81, 10, 45, 29, 79, 98};
```

isso significa que **vetor[0]** terá o valor 17 e **vetor[9]** terá o valor 98

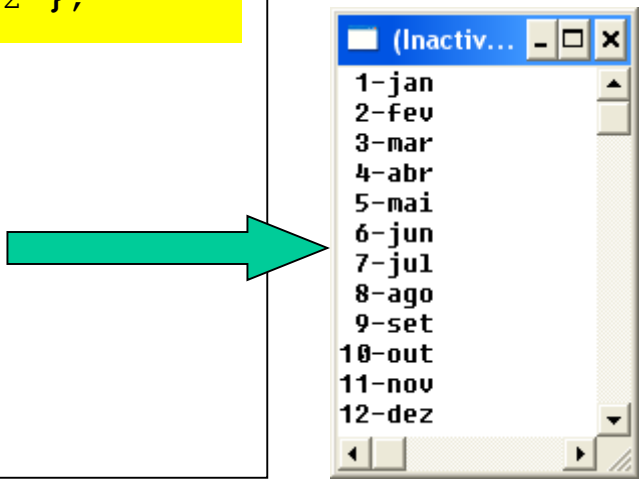
```
int matriz[2][4] = {17, 33, 21, 15,
                    13, 81, 97, 67};
```

nos vetores bidimensionais os valores são inicializados por linha, no exemplo, **matriz[1][2]** (segunda linha e terceira coluna) terá o valor 97

Vetor de Strings

Para criar um vetor de strings, usa-se uma matriz bidimensional de caracteres. Na declaração o tamanho do índice esquerdo indica o número de strings e o "tamanho do índice direito especifica o comprimento máximo de cada string".

```
#include "stdio.h"

void main() {

  char nomeMes[12][3] = {"jan", "fev", "mar", "abr", "mai",
    "jun", "jul", "ago", "set", "out", "nov", "dez"};

  for (int i=0; i<12; i++) {
    printf("%2d-", (i+1));

    for (int j=0; j<3; j++)
      printf("%c", nomeMes[i][j]);

    printf("\n");
  }
}
```



Vetores Multidimensionais

C permite vetores com mais de duas dimensões. O limite exato, se existe, é determinado pelo compilador usado.

A forma geral da declaração de um vetor multidimensional segue a seguinte sintaxe:

tipo nome[Tamanho1][Tamanho2][Tamanho3]...[TamanhoN];

por exemplo, **int** x[2][2][4]; onde "x" representa uma estrutura de dados com 2 * 2 * 4 = 16 números inteiros, indexados da seguinte forma:

x[0][0][0]	x[0][1][0]	x[1][0][0]	x[1][1][0]
x[0][0][1]	x[0][1][1]	x[1][0][1]	x[1][1][1]
x[0][0][2]	x[0][1][2]	x[1][0][2]	x[1][1][2]
x[0][0][3]	x[0][1][3]	x[1][0][3]	x[1][1][3]

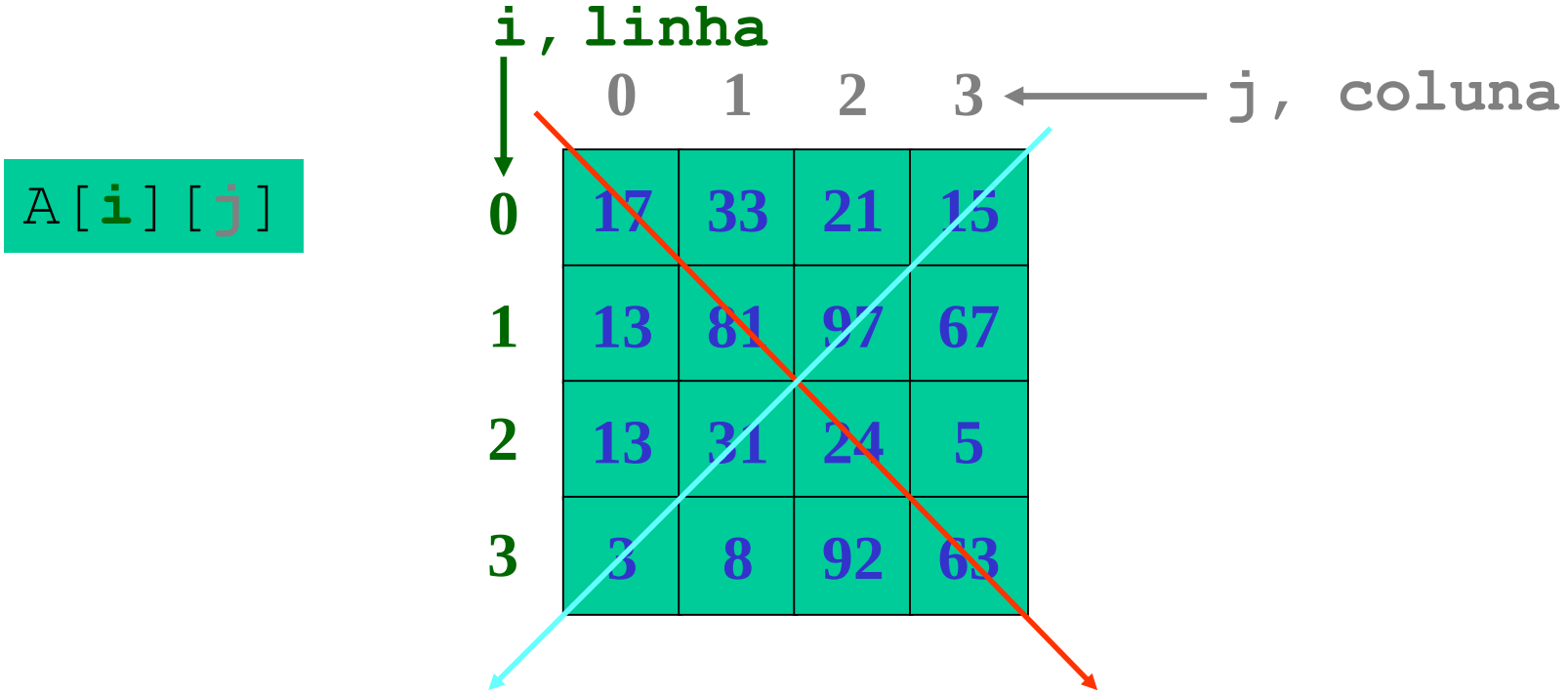
Conceitos sobre Matrizes

- Matriz quadrada
  - número de linhas igual ao número de colunas
- Diagonal principal
  - $A_{i,j}$ para todo i == j
- Diagonal secundária
  - $A_{i,j}$ para todo (i + j) == (n - 1)

- Matriz esparsa
  - a maioria dos elementos da matriz é zero

Exemplo de **Matriz Quadrada**

```
const n = 4;
int A[n][n] = {17, 33, 21, 15, 13, 81, 97, 67,
               13, 31, 24,  5,  3,  8, 92, 63};
```




```
diagonal secundária [(i+j)==(n-1)]:
A[0][3]       15
A[1][2]       97
A[2][1]       31
A[3][0]       3
```

```
diagonal principal (i==j):
A[0][0]       17
A[1][1]       81
A[2][2]       24
A[3][3]       63
```

```
// Dado um vetor A de duas dimensões com 4 linhas e 4 colunas. Elaborar um programa que
// calcule e escreva o valor da soma dos elementos deste vetor, ou matriz.

#include "stdio.h"
#include "stdlib.h"
#include "conio.h"

void main() {
// declaração da estrutura de dados
  const n = 4;
  int A[n][n]; // matriz quadrada de n linhas x n colunas
  int i, j, sm = 0;

// entrada- gerar nros aleatórios para armazenar no vetor bidimensional
  randomize();
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      A[i][j] = random(10);

// processamento: somando todos os elementos da matriz
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      sm = sm + A[i][j];

// saída: imprime os resultados
  clrscr();
  printf("Matriz A\n");
  printf("==============\n");
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
      printf("%2d ", A[i][j]);
    printf("\n");
  }

  printf("\nA soma dos elementos da matriz é igual a %d", sm);
}
```

```
Alimentando a matriz por linha:
i:0,       1,       2,       3,
j:0,1,2,3  0,1,2,3  0,1,2,3  0,1,2,3
```