departamento de informática

engenharia informática

Algoritmos e Programação

1º Ano - 1º Semestre

2. Programação numa linguagem de alto nível (C) – Parte 3

Escola Superior de Tecnologia e Gestão de Viseu

Agradecimentos a Francisco Morgado, Carlos Simões e Jorge Loureiro

2. Programação numa linguagem de alto nível (C) - Part 3

2.1 Introdução

2.2 Estrutura de um programa em linguagem C

2.3 Tipos de dados básicos

2.4 Testes e condições - expressões e operadores. Precedências

2.5 Estruturas de repetição

2.6 Funções

2.7 Tipos de dados estruturados: vetores e strings

2.8 Apontadores (Pointers)

2.9 Passagem de parâmetros de tipos estruturados

2.10 Estruturas

2.11 Memória dinâmica

2.7 Tipos de dados estruturados: arrays e strings

Arrays

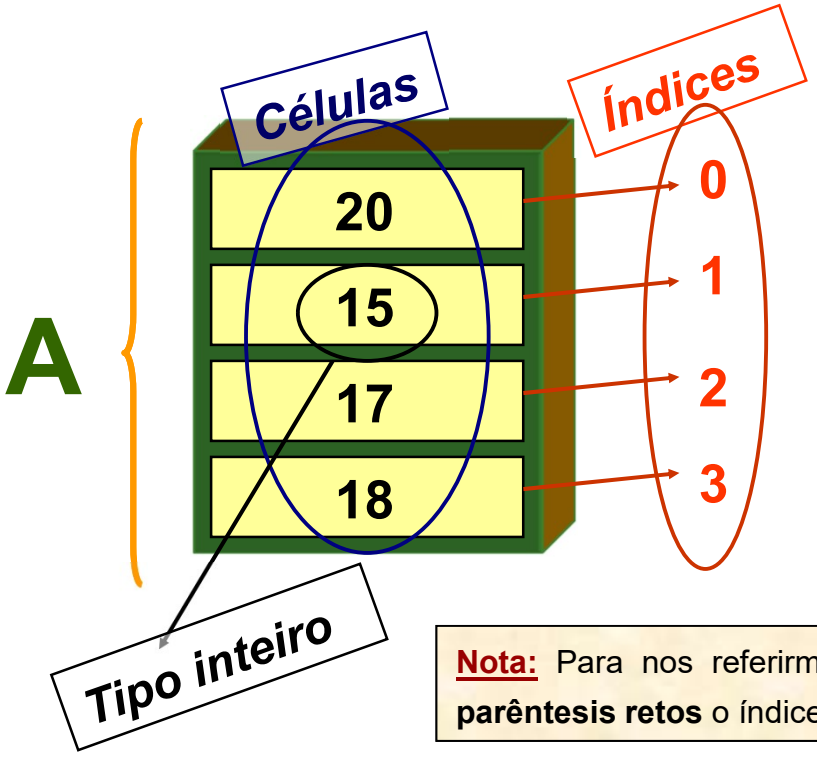
- Conceito e utilidade
- Declaração
- Definição de constantes em C
- Inicialização; índices
- Passagem de vetores para funções
- Operações básicas com vetores (*arrays unidimensionais)*
- *Arrays* multidimensionais
- Declaração e inicialização
- Operações básicas com matrizes (*arrays bidimensionais)*

Strings

- Caracteres e *strings*
- Observações relativas à representação de caracteres
- Escrita e leitura de *strings* e caracteres
- Passagem de *strings* para funções

Conceito e utilidade

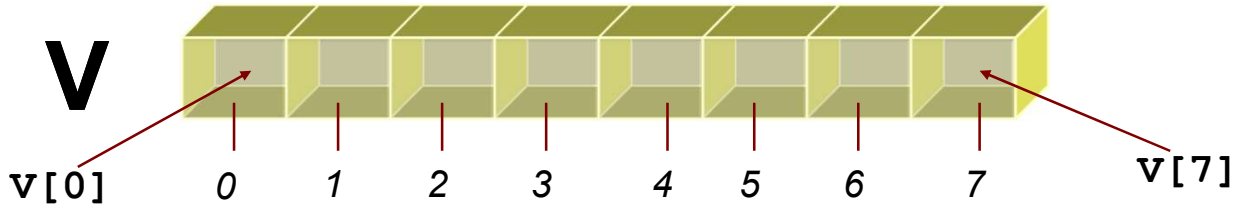
Por vezes torna-se necessário/conveniente referenciar uma zona de memória onde estão guardados vários valores de um mesmo tipo de dados, utilizando um mesmo nome. Para o efeito, utilizam-se variáveis do tipo estruturado: ***Arrays***.



Array - trata-se de um tipo de variável estruturada, constituída por um conjunto de "células", identificadas univocamente por um "índice" (endereço), cujo conteúdo é constituído por um valor de determinado tipo.

Nota: Para nos referirmos a cada um dos seus valores, utilizamos o nome do **Array** e **entre parêntesis retos** o índice da célula onde se encontra o valor.

Arrays Unidimensionais (Vetores)




```
int V [8]
{ Array unidimensional c/ 8 células, com valores inteiros }
```

Declaração de *Arrays* em C

SINTAXE
tipo identificador **[número de elementos]**

Tipo → estabelece o tipo de dados de cada componente do *array*

Identificador → nome pelo qual o *array* vai ser conhecido

Número de elementos → valor constante que indica quantos elementos tem o *array*

- **Em C os índices de um *array* com *n* elementos variam sempre entre 0 e *n*-1**

- **O índice da primeira componente é 0 (zero) e o da *n*-ésima é *n*-1**

Na declaração

```
int notaAP[100]; // notaAP é um vetor com 100 inteiros

int             - tipo de cada um dos elementos do vetor
notaAP          - nome do vetor
100             - número de elementos do vetor
notaAP[i]       - aquilo que está na i-ésima componente do vetor
```

Definição de constantes em C

As constantes devem ser declaradas fora das funções, por forma a serem «visíveis» por todo o código do programa. Geralmente a sua definição é feita logo a seguir às linhas dos *#include*

A definição de constantes pode ser realizada de duas formas distintas:

- **Através da palavra reservada const**

  const tipo símbolo = valor;

- **Através da diretiva de pré-processamento #define**

  #define símbolo valor

Diferenças entre const e #define

- Uma constante definida com const existe fisicamente numa determinada posição de memória. Fica com o tipo que lhe foi colocado na definição
- Uma constante definida com #define não existe fisicamente em memória, sendo o seu valor substituído ao longo do programa na fase de pré-processamento (ainda antes da compilação)
- Enquanto que const é uma palavra reservada do C, #define é uma diretiva que indica ao pré-processador que o símbolo que se lhe segue vai ficar com o valor que aparece a seguir ao símbolo. O tipo associado à constante é o que resulta da expressão que aparece na componente valor
- Uma vez que #define não faz parte da linguagem C, a respetiva linha de código não é seguida de ;

Inicialização. Índices.

- É possível inicializar as componentes de um *array* logo na declaração.

→ **tipo** identificador [n] = {valor$_1$,…, valor$_n$}

Exemplo

```
char vogal[5] = {'a', 'e', 'i', 'o', 'u'};
```

⇔

```
char vogal[5];
vogal[0]='a';
vogal[1]='e';
vogal[2]='i';
vogal[3]='o';
vogal[4]='u';
```

- Se um *array* for declarado com n elementos e apenas k (k<n) forem inicializados, os restantes ficam inicializados com o valor 0 (ESPAÇO - caracter nº zero - tratando-se do tipo char) .

Exemplo

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

⇔

```
int v[5] = {10,15,20,0,0};
```

- **As posições de um *array* só são inicializadas se a declaração for seguida de = {…}**

- NOTA: O compilador não deteta se os índices utilizados estão fora da gama declarada.

Passagem de *arrays* unidimensionais para funções

Em C, dentro de uma função, não é possível saber com quantos elementos foi declarado um *array* passado como argumento para essa função. A dimensão a considerar é da exclusiva responsabilidade do programador.

Exemplo

```
#include <stdio.h>
void inicializa (int s[ ], int n)
{
   int i;
   for (i=0; i<=n-1; i++)
        s[i] = i;
}
main()
{
   int x[15], y[20];
   inicializa(x, 15);
   inicializa(y, 20);
   printf("\n");
}
```

Qual o valor das componentes do vetor x e do vetor y imediatamente antes de o programa terminar?

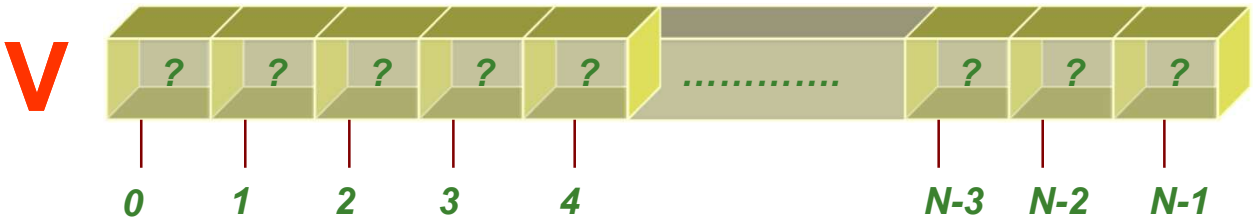
Operações básicas com vetores – arrays unidimensionais

- Ler as componentes de um vetor
- Escrever o valor de cada componente de um vetor
- Máximo e mínimo e respetiva posição
- Soma de todas as componentes
- Média aritmética das componentes
- Troca de duas componentes
- Inversão
- Permutação circular
- Remoção de uma componente
- Inserção de uma nova componente

OBSERVAÇÃO

Nas funções apresentadas seguidamente pressupõe-se que no programa principal foi declarado um *array* unidimensional `v` com N componentes inteiras.

Ler as componentes de um vetor



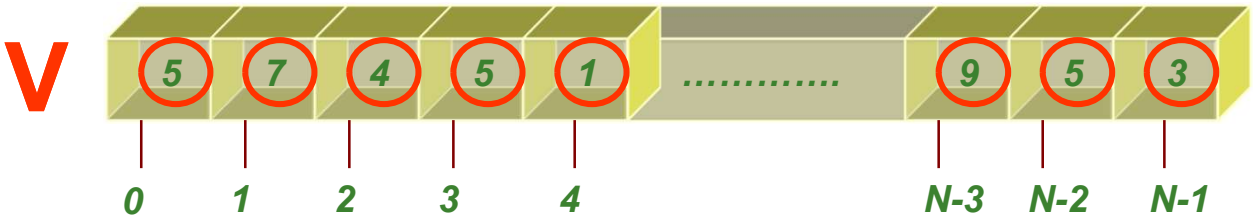
Descrição Algorítmica

```
...
PARA índice DESDE 0 ATÉ N-1 FAZ
   Escrever (“Insira o elemento”, índice+1, ”: ”);
   Ler ( v( índice ) );
FIM-PARA
...
```

Codificação em C

```
void leVetor(int x[], int tam)
// Funçao para leitura de um vetor com tam inteiros
{
   int i;
   for (i = 0; i < tam; i++)
   {
      printf("Qual o %d° elemento? ", i+1);
      scanf("%d",&x[i]);
   }
}
```

Escrever o valor de cada componente de um vetor

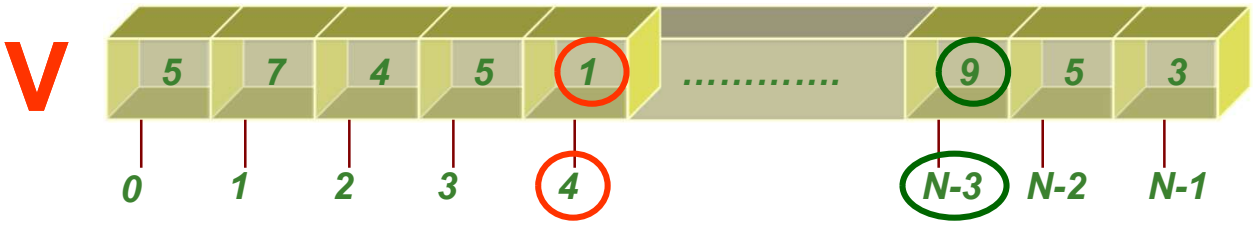


Descrição Algorítmica → ***Codificação em C***

```
...
PARA índice DESDE 0 ATÉ N-1 FAZ
  Escrever ("v(", índice+1,")= ",V( índice ));
FIM-PARA
...
```

```
void escreveVetor(int x[], int tam)
//Funçao para escrita de um vetor com tam inteiros
{
  int i;
  for (i = 0; i < tam; i++)
    printf("\nx[%d](%dª) = %d   ", i, i+1, x[i]);
  printf("\n");
}
```

Máximo de um vetor, devolvendo a posição do elemento respetivo.



Descrição Algorítmica → **Codificação em C**

```
...
Maximo ← v(0);
PosMax ← 0:
PARA indice DESDE 1 ATÉ N-1 FAZ
   SE v(índice) > Máximo ENTÃO
      Maximo ← v(indice);
      PosMax ←indice;
   FIM-SE
FIM-PARA
Escrever("Valor máximo:", Máximo," posição:", PosMax);
...
```

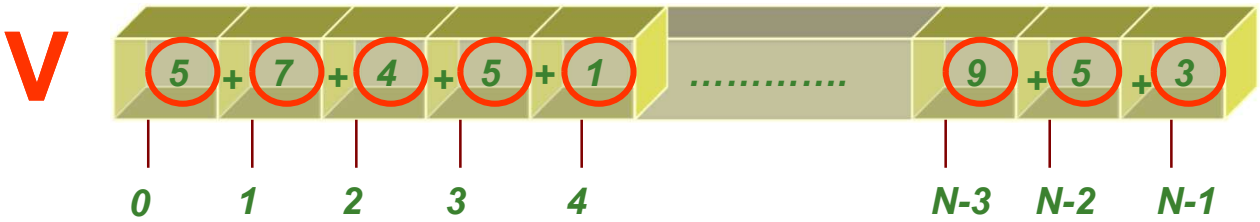
```
int posMax (int x[], int tamanho)
 // Função para determinar o índice da MAIOR
 // componente de um vector
{
  int p, i;
  p = 0;
  for (i = 1; i < tamanho; i++)
    if (x[i] > x[p])
      p = i;
  return p;
}
```

Que acontecerá se colocar ; a seguir à condição do if?
E se estiver a seguir ao for, antes do if?

Exercícios

1. Escrever um programa que invoque a função *posMax*
2. Escrever uma função para determinar a posição do mínimo

Soma das componentes de um vetor



Descrição Algorítmica

```
...
soma ← 0;
PARA índice DESDE 0 ATÉ N-1 FAZ
   soma ← soma + v( índice ) ;
FIM-PARA
...
```

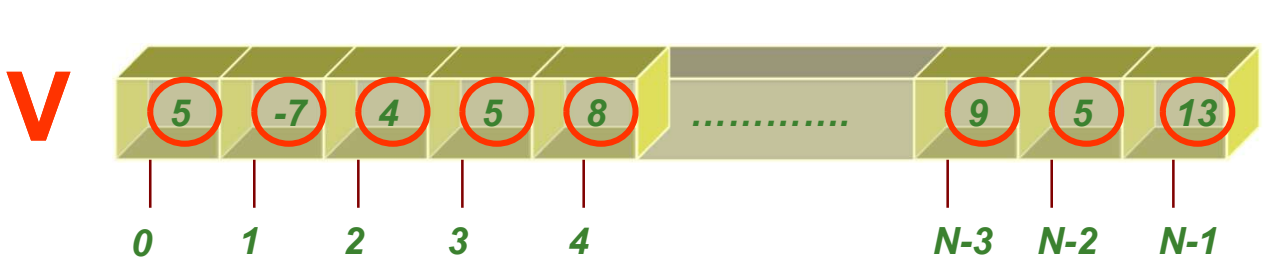
Codificação em C

```
int somaComponentes(int x[], int tamanho)
//Funçao que calcula a soma das componentes de um vector
{
  int soma, i;
  soma = 0;
  for (i = 0; i < tamanho; i++)
       soma += x[i];
  return soma;
}
```

Chamada

```
printf("\nSoma das componentes: %d", somaComponentes(v, dim) );
```

Média aritmética de todos os elementos de um vetor



$$\overline{V} = \frac{\sum_{i=1}^{N} V(i)}{N}$$

Descrição Algorítmica → **Codificação em C**

```
...
soma ← 0;
PARA índice DESDE 0 ATÉ N-1 FAZ
   soma ← soma + v( índice ) ;
FIM-PARA
Média ← soma/ N ;
...
```

```
float mediaComponentes(int x[], int tamanho)
//Funçao que calcula a soma das componentes de um vector
{
  int soma, i;
  soma = 0;
  for (i = 0; i < tamanho; i++)
       soma += x[i];
  return (float)soma/tamanho;
}
```

(casting)

Chamada

```
printf("\n Media das componentes: %0.2f", mediaComponentes(v, dim) );
```

(Casting)

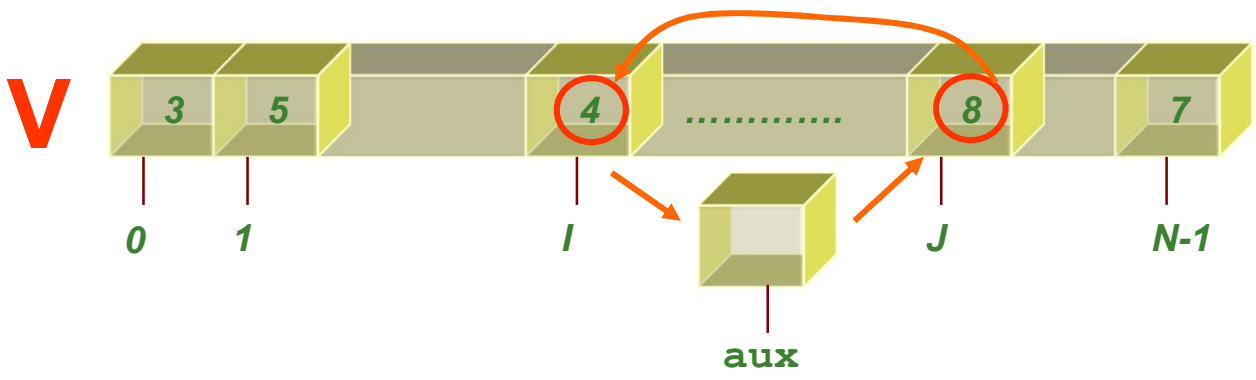
Sempre que, numa variável/expressão, queremos modificar temporariamente o tipo do respetivo valor, podemos fazê-lo, indicando o tipo pretendido entre parêntesis antes da variável/expressão.

Exemplo

Escrever uma função que, dado um número real, mostre a sua parte inteira e a sua parte fracionária

```
void decompoeReal(float x)
//Funçao que mostra a parte real e a parte fracionária de um n° real
{
  printf("Parte inteira de %f: %d\n", x, (int)x);
  printf("Parte real de %f: %f\n", x, x-(int)x);
}
```

Troca de dois elementos de um vetor



Descrição Algorítmica

```
...
Escrever("Qual o índice do 1.º elemento a trocar?");
Ler ( i );
Escrever("Qual o índice do 2.º elemento a trocar?");
Ler ( j );
Aux ← v( i );
v( i ) ← v( j );
v( j ) ← Aux;
...
```

Codificação em C

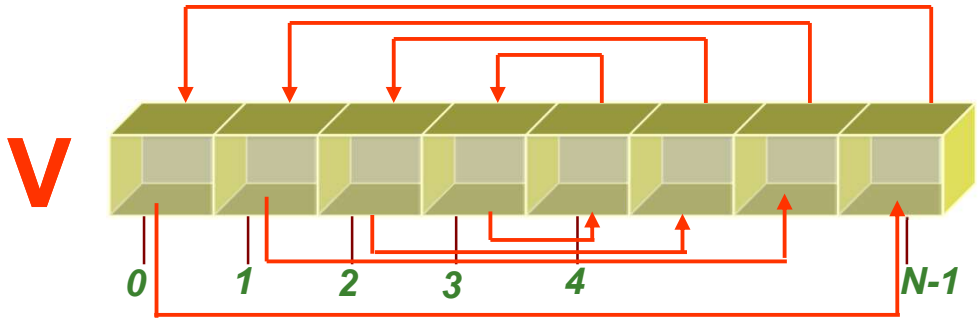
Procedimento

```
void troca2componentes(int x[], int p1, int p2)
// Funçao para troca das componentes nas posicoes p1 e p2
{
        int aux;
        aux   = x[p1];
        x[p1] = x[p2];
        x[p2] = aux;
}
```

Exercício

Chamar esta função num programa que leia e escreva vetores

Inversão de um vetor

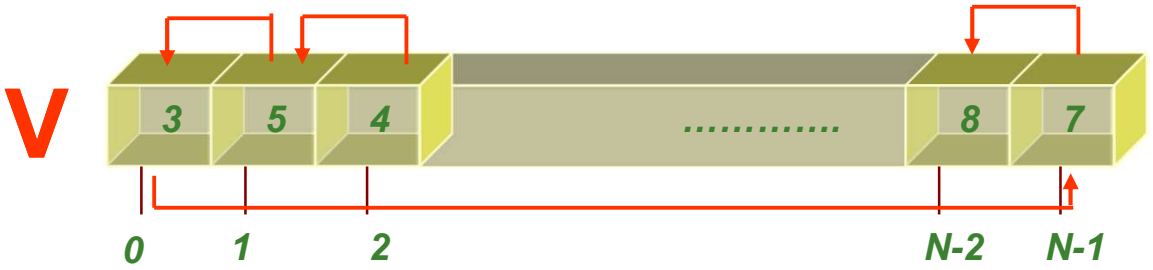


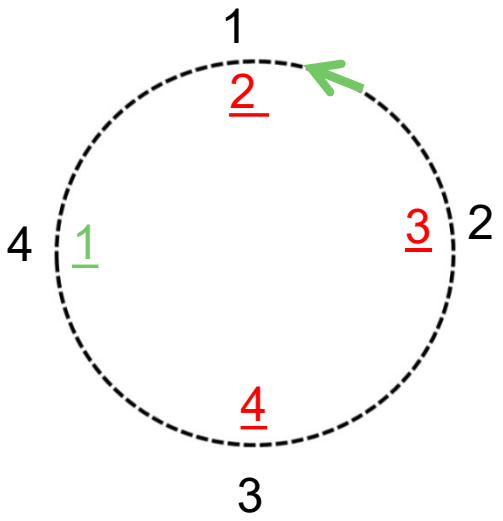
Descrição Algorítmica → ***Codificação em C***

```
...

k ←N \ 2-1; { Divisão inteira }
PARA índice DESDE 0 ATÉ k FAZ
   Aux ← v( índice ) ;
   v( índice ) ← v( N – índice - 1 );
   v( N – índice - 1 ) ← Aux;
FIM-PARA

...
```

```
void inverteVetor(int x[], int tam)
 // Funçao para inverter um vetor
{
        int i, k, aux;
        k = tam/2;
        for (i=0; i<k ; i++)
        {
                aux = x[i];
                x[i] = x[tam-1 -i];
                x[tam-1 -i] = aux;
        }
}
```

Permutação circular de um vetor



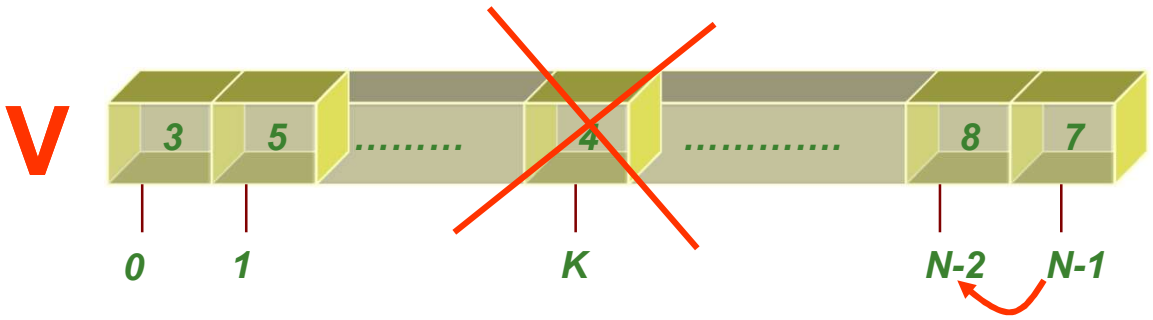


Descrição Algorítmica → ***Codificação em C***

```
...
aux ← v( 0);
PARA índice DESDE 1 ATÉ N -1 FAZ
    v( índice - 1 ) ← v( índice );
FIM-PARA
v( N-1 ) ← aux;
...
```

```
void permutacaoCircular(int x[], int tam)
//Permuta circularmente as componentes de um vetor
{
     int i, aux;
     aux = x[0];
     for (i=0; i<tam-1 ; i++)
        x[i] = x[i + 1];
     x[tam-1] = aux;
}
```

Remoção de elementos de um vetor



Descrição Algorítmica → ***Codificação em C***

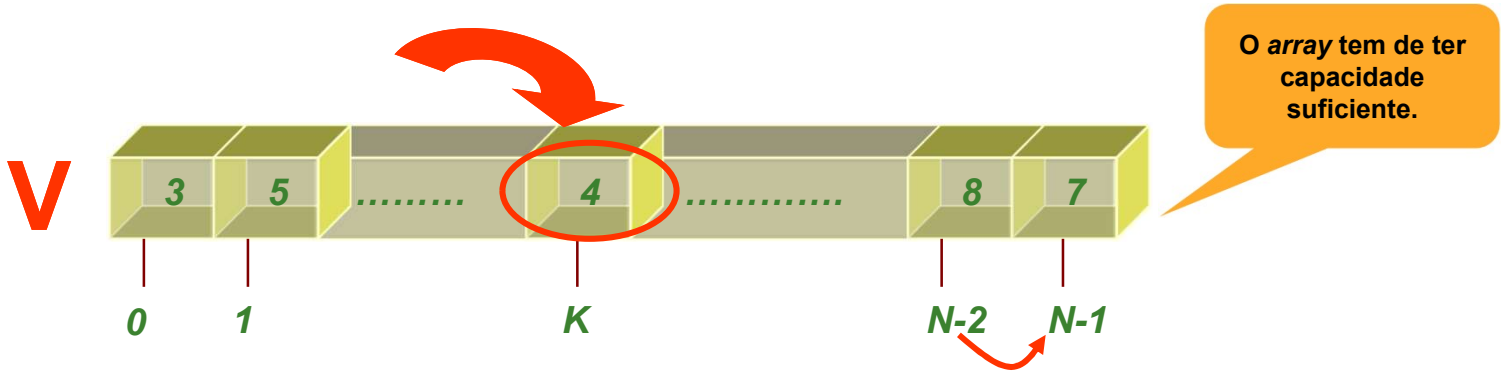
```
...
Escrever("Qual o índice do elemento a remover?");
Ler ( k );
PARA índice DESDE k ATÉ N - 2 FAZ
    v( índice ) ← v( índice + 1 )
FIM-PARA
N ← N – 1;
...
```

Atenção aos valores na chamada!

```
void removeComponente (int x[], int tam, int k)
{
      int i;
      for (i=k; i<tam-1 ; i++)
                  x[i] = x[i+1];
}
```

```
...
removeComponente(v, dim, 1);
printf("\n Removida a componente da 2a posição");
...
```

Inserção de um novo elemento no vetor



Descrição Algorítmica → *Codificação em C*

```
...
Escrever("Qual o índice do elemento a inserir?");
Ler ( k );
Escrever("Qual o valor a colocar nesse elemento?")
Ler ( val );
PARA índice DESDE N-1 ATÉ k FAZ (Salto –1)
    v( índice ) ← v( índice – 1 );
FIM-PARA
v( índice ) ← val;;
...
```

```
void insereComponente(int x[], int tam, int k, int valor)
{
      int i;
      for (i=tam-1; i>k ; i--)
                  x[i] = x[i-1];
      x[k] = valor;
}
```

```
...
insereComponente(v, dim++, 1, 24);
printf("\n Inserida nova componente com o valor 24, na 2a posição");
...
```

Exercícios: indicar os erros de sintaxe ou semântica existentes nos seguintes excertos de programa

a)

```
...
{       int x[20], i;

        for (i=1; i<=20; i++)
                x[i] = 0;
}
```

b)

```
...
{       int x[20], i;

        for (i=0; i<20; i++)
                x[i] = 0;
        x[i] = 123;
}
```

c)

```
...
{       int i=20;
        int x[i];
        int y[ ];

        for (i=0; i<20; i++)
            {       x[i] = 0;
                    y[i] = 1;
            }
}
```

d)

```
...
#define MAX 20
void main()
{
        int x[MAX];

        for (i=0; i<MAX; i++)
            x[i] = 0;
        x[i] = 456;
}
```

e)

```
...
{       int i;
        int x[5] = {1, 2, 3, 4, 5, 6};

        for (i=0; i<5; i++)
            x[i] = i;
        x[i] = 234;
}
```

respostas

a) **Os índices do vetor variam entre 0 e 19**

b) **Depois de terminado o ciclo, i tem o valor 20**

c) **A dimensão de um vetor tem de ser uma constante**
Não se pode declarar um vetor sem dimensão

d) **Todas as ocorrências de MAX no programa são substituídas por 20; Depois de terminado o ciclo, i tem o valor 20.**

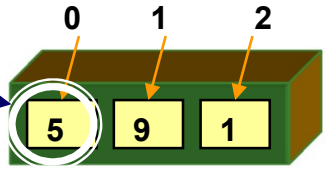
e) **O vetor tem apenas 5 componentes. Depois de terminado o ciclo, i tem o valor 5.**

Arrays multidimensionais

SINTAXE
tipo identificador **[dim$_1$] [dim$_2$] … [dim$_n$];**

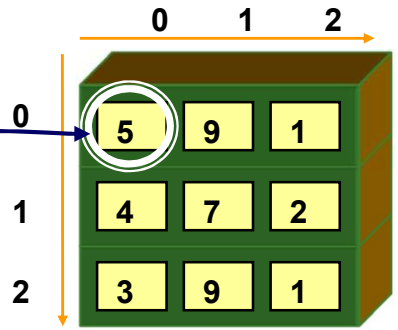
Array uni-dimensional - (1-D)

```
int UniArray [3]
UniArray[0] = 5;
```



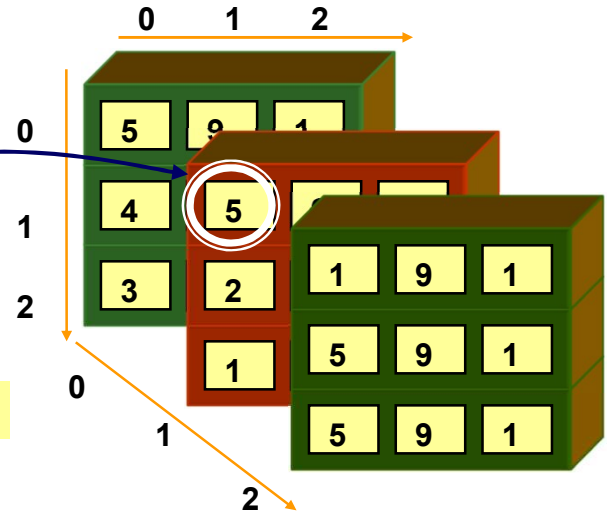
Array bi-dimensional - (2-D)

```
int BiArray [3] [3]
BiArray[0][0] = 5;
```



Array tri-dimensional - (3-D)

```
int TriArray [3][3][3]
TriArray[0][0][1] = 5;
```



- É possível inicializar as componentes de um *array* multidimensional logo na declaração.

Exemplo

```
int v[2] [3] = {{1, 2, 3}, {4, 5, 6}};
```

Operações básicas com arrays bidimensionais (matrizes)

- Ler uma matriz
- Escrever uma matriz
- Determinar a transposta de uma matriz
- Determinar o valor máximo numa matriz com nl linhas e nc colunas
- Verificar se uma dada matriz é ou não simétrica
- Calcular o traço de uma matriz
- Adição/Subtração de matrizes

OBSERVAÇÃO

Nas funções apresentadas, correspondentes a estas operações básicas, considera-se declarado um ARRAY bidimensional de INTEIROS (int mat[MAX][MAX]), a que corresponderá o parâmetro formal **matriz**. Aos parâmetros formais **tamL** e **tamC** deverão corresponder parâmetros reais do tipo inteiro, com valor não superior a **MAX**.

Passagem de *arrays* multidimensionais para funções

- A passagem de *arrays* com mais de uma dimensão para uma função realiza-se indicando no cabeçalho, obrigatoriamente, o número de elementos da cada uma das dimensões (a dimensão mais à esquerda pode ser omitida, colocando apenas [ ] ou *)

Ler uma matriz

```
void leMatriz(int matriz[MAX][MAX], int tamL, int tamC)
 // Funçao para leitura de uma matriz tamLxtamC
{
     int i,j;
     for (i = 0; i < tamL; i++)
         for (j = 0; j < tamC; j++)
         {
              printf("\n(%d,%d])= ", i+1, j+1);
              scanf("%d",&matriz[i][j]);
         }
}
```

Faça uma simulação para 3x3

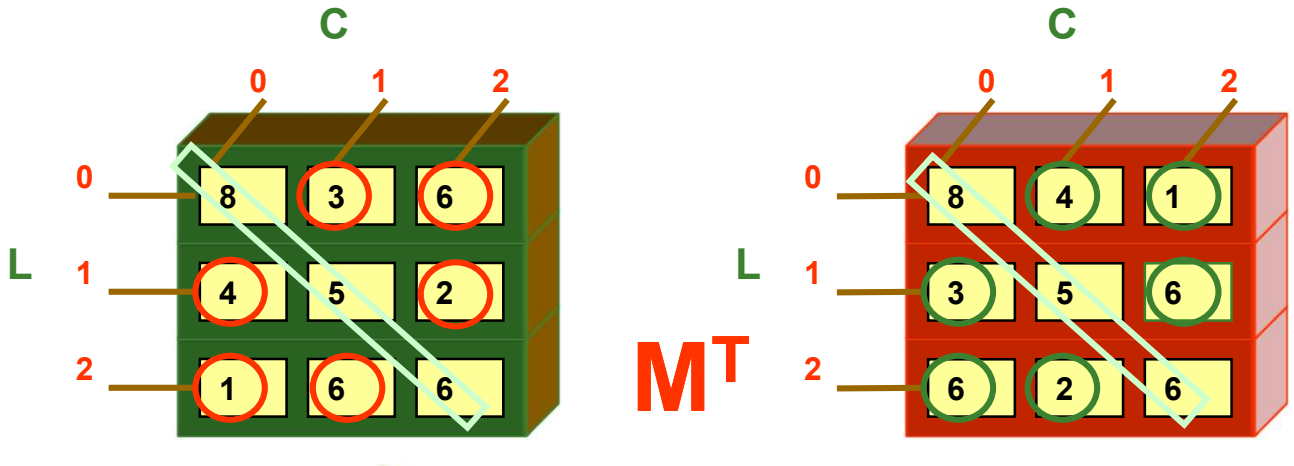
Escrever uma matriz

Faça uma simulação para a matriz identidade 3x3

```
void escreveMatriz(int matriz[MAX][MAX], int tamL, int tamC)
 // Funçao para escrita de uma matriz tamLxtamC
{
     int i,j;
     for (i = 0; i < tamL; i++)
     {
        for (j = 0; j < tamC; j++)
            printf("%5c[%d,%d]=%d",' ', i+1, j+1, matriz[i][j]);
        printf("\n");
     }
}
```

Transposta de uma matriz

$M^T(c,l) = M(l,c)$



Descrição Algorítmica

```
...
PARA L DESDE 1 ATÉ Tot_L FAZ
    PARA C DESDE 1 ATÉ Tot_C FAZ
        M_T( L, C ) ← M( C, L )
    FIM-PARA
FIM-PARA
...
```

A matriz pode não ser quadrada.

Determinar a transposta de uma matriz

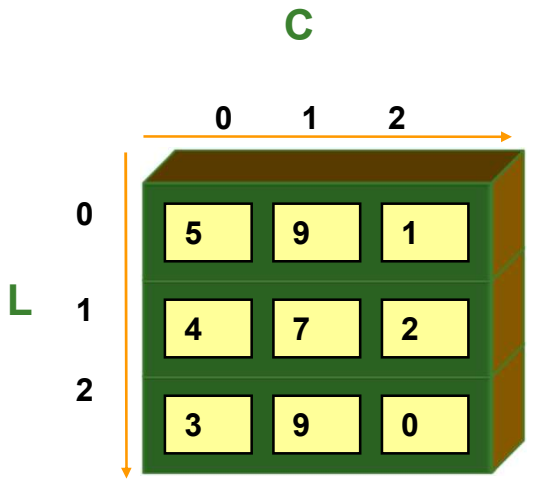
Codificação em C

```
void transpor (int matriz[MAX][MAX], int tam)
// transpõe a matriz m com tam por tam elementos
{
     int i, j, tmp;
     for (i=1; i<tam; i++)
          for (j=0; j<i; j++)
               {
                    tmp      = m[i][j];
                    m[i][j] = m[j][i];
                    m[j][i] = tmp;
               }
}
```

```
leMatriz(mat1, dim, dim);
transpor(mat1, dim);
printf("\n  MATRIZ TRANSPOSTA \n");
escreveMatriz(mat1, dim, dim);
```

Chamada

Faça uma simulação para esta matriz



Determinar o valor máximo numa matriz com nl linhas e nc colunas

```
int maiorElemento(int matriz[MAX][MAX], int tamL, int tamC)
{
   int i, j;
   int maior = matriz[0][0];
   for (i = 0; i < tamL; i++)
   {
      for (j = 0; j < tamC; j++)
         if (matriz[i][j] > maior)
            maior = matriz[i][j];
   }
   return maior;
}
```

Chamada

```
printf("\n Valor maximo da matriz: %d\n", maiorElemento(mat, vnl, vnc));
```

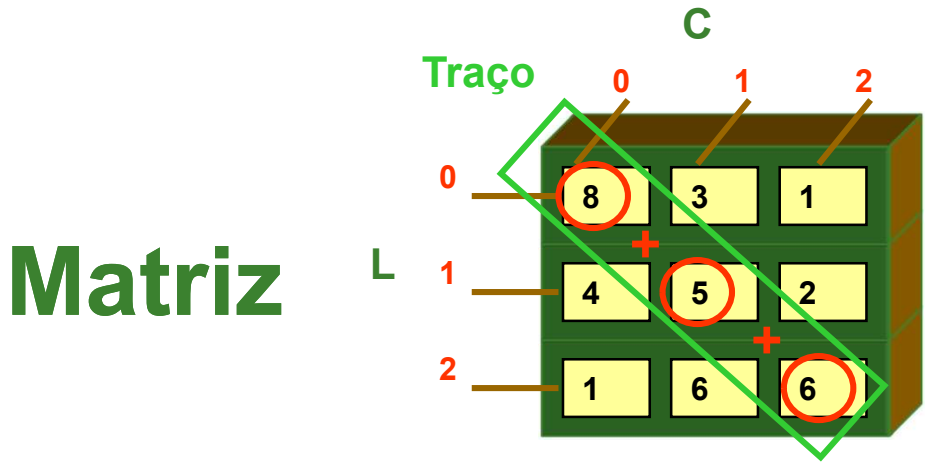
Verificar se uma dada matriz é ou não simétrica

```
int MatrizSimetrica(int matriz[MAX][MAX], int tamL, int tamC)
{
   int i, j, teste;
   teste = (tamL==tamC);
   if (teste)
      for (i = 1; i < tamL; i++)
      {
         for (j = 0; j < i; j++)
            if (matriz[i][j] != matriz[j][i])
                return 0;
      }
   return teste;
}
```

```
if (MatrizSimetrica(mat, vnl,vnc))
        printf("\n Matriz simetrica\n");
else printf("\n Matriz NAO simetrica\n");
```

Chamada

Traço de uma matriz



Descrição Algorítmica

```
...
SE N_L = N_C ENTÃO
    Traco ← 0;
    PARA índice DESDE 0 ATÉ N_L-1 FAZ
          Traco ← Traco + Matriz(índice, índice);
    FIM-PARA
    Escrever("O traço da matriz é: ", Traco)
SENÃO
    Escrever("Erro: A matriz não é válida")
FIM-SE
...
```

Calcular o traço de uma matriz

Codificação em C

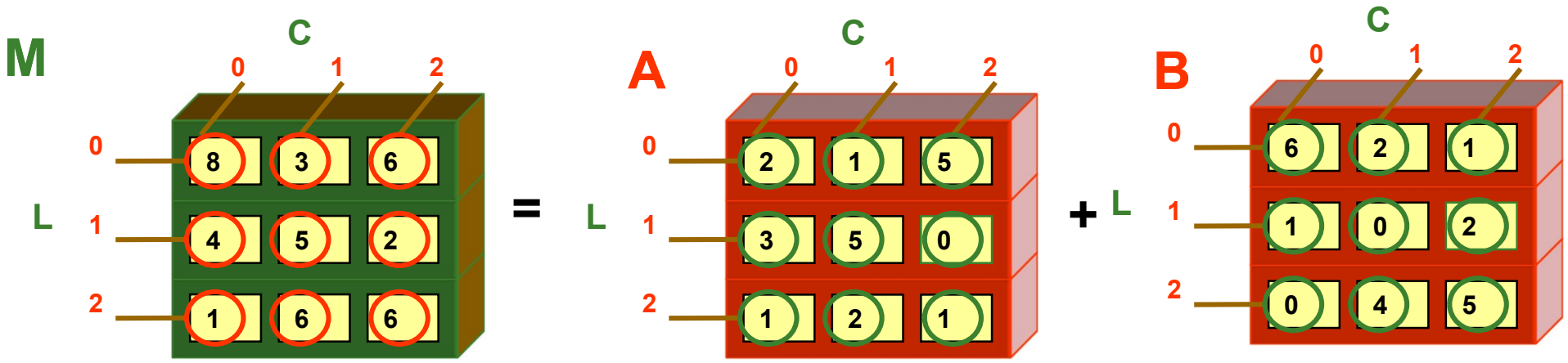
```
//chamada só para matrizes quadradas (tam x tam)
int traco(int matriz[MAX][MAX], int tam)
{
   int i, traco;
   traco = 0;
   for (i = 0; i< tam; i++)
      traco += matriz[i][i];
   return traco;
}
```

```
if (vnl==vnc)
        printf("\n Traco= %d",traco(mat, vnl));
else printf("\n Matriz NAO simetrica\n");
```

Chamada

Adição/Subtração de Matrizes

$$M(l,c) = A(l,c) \pm B(l,c)$$



Descrição Algorítmica

```
...
PARA L DESDE 1 ATÉ Tot_L FAZ
    PARA C DESDE 1 ATÉ Tot_C FAZ
        M(L, C) ← A(L, C) + B(L, C);
    FIM-PARA
FIM-PARA
...
```

Determinar a (matriz) soma de duas matrizes da mesma dimensão

Codificação em C

```
void somaMatrizes(int ma[MAX][MAX], int mb[MAX][MAX], int mc[MAX][MAX],int tamL,int tamC)
// determina a matriz mc, soma de duas matrizes dadas, ma e mb
{
    int i, j;
    for (i=0; i<tamL; i++)
        for (j=0; j<tamC; j++)

            mc[i][j] = ma[i][j]+ mb[i][j];
}
```

No caso da subtração,
basta trocar o sinal para "–".

```
leMatriz(mat1, dim, dim);
leMatriz(mat2, dim, dim);
somaMatrizes(mat1, mat2, mat3, dim,dim);
printf("\n  MATRIZ SOMA\n");
escreveMatriz(mat3, dim, dim);
```

Chamada

Caracteres versus *strings*

Uma *string* é um conjunto de caracteres armazenados num vetor.

Em C as *strings* são representadas utilizando aspas (“), enquanto que os caracteres são representados entre plicas (‘).

Exemplos de *strings*

```
“Algoritmos e Programação”
“Manuel”
“M”
```

Exemplos de caracteres

```
‘M’
‘<’
‘*’
```

NOTA

‘M’ ocupa 1 Byte
“M” ocupa 2 Bytes

Um vetor de caracteres pode não ser uma *string*

Uma *string* corresponde sempre a um vetor de caracteres com um caracter especial como terminador (‘\0’)

```
char s[]= “aeiou”; - neste tipo de inicializações, é colocado automaticamente
                     o caracter terminador

char vogais[]={‘a’,‘e’,‘i’,‘o’,‘u’};  – não é uma string: não é colocado
                                        o caracter terminador
```

Observações relativas à representação de caracteres

Alguns caracteres não são passíveis de ser escritos através do teclado. Para os representarmos, usamos uma combinação de caracteres iniciada com \ (Backslash), indicando assim que o caracter seguinte deve ser entendido de forma especial.

Exemplos

Tabulação horizontal -> \t
Backspace -> \b
Representação do caracter \ -> \\
Fim de string -> \0
Mm em hexadecimal -> \xMm

Caracter ‘ (plica) -> \’
Caracter ?(ponto de interrogação) -> \?
Mudança de linha -> \n
Caracter % -> %%
Caracter “ (aspas) -> \”

Diversas formas de atribuição de valor a uma variável do tipo CARACTER

Exemplo: colocar ‘A’ na variável c, do tipo CHAR

```
c = ‘A’; // formato tradicional
c = 65; // caracter cujo código ASCII é 65
c = ‘\101’; // caracter cujo código ASCII em octal é 101
c = ‘\x41’; // caracter cujo código ASCII em hexadecimal é 41
```

Escrita e leitura de strings e caracteres

Escrita de strings

A escrita de *strings* é realizada por duas funções distintas: printf e puts

Função **printf**

A função **printf** recebe como formato uma *string*, que pode ser escrita diretamente

```
printf("Bom dia\n");
```

Contudo, uma *string* também pode ser escrita como qualquer outra variável:

```
char d[50] = "ALGORITMOS E PROGRAMAÇÃO";
printf("Unidade Curricular: %s\n", d);
```

Função **puts**

A função **puts** permite, unicamente, a escrita de *strings*, sejam elas constantes ou estejam armazenadas em variáveis.

printf("Boa tarde\n"); ⟺ **puts("Boa tarde");**

A função *puts* coloca no ecrã a *string* passada à função e em seguida faz uma mudança de linha.

Leitura de strings

scanf versus gets

Função **scanf**

A função **scanf** permite realizar a leitura de *strings* através do formato %s. No entanto, a variável que recebe a *string* NÃO É PRECEDIDA de &, ao contrário do que acontece com todos os outros tipos de variáveis enviadas para o **scanf**.

NOTA

A função **scanf** realiza a leitura de uma só palavra (termina a leitura ao encontrar um <ESPAÇO>, <TAB> ou <ENTER>)

Função **gets**

A função **gets** permite colocar na variável que recebe como parâmetro todos os caracteres introduzidos pelo utilizador

```
#include <stdio.h>
main()
{
        char nome[60];
        printf("\nInsira o nome completo: ");
        gets(nome);
        printf("\nNome completo: %s\n", nome);
}
```

Notas sobre leitura e escrita de caracteres

getchar versus scanf

Enquanto que a função **scanf** é uma função genérica de leitura, a função **getchar** é especificamente dedicada à leitura de caracteres.

EXEMPLO: Sendo *letra* uma variável do tipo **char,** a instrução

letra = **getchar**();

é equivalente a

scanf (" %c", &letra);

Função **putchar**

A função **putchar** recebe um caracter e coloca-o no ecrã

EXEMPLO: Sendo letra uma variável do tipo **char**, tem-se

putchar(letra); ⟺ **printf**("%c", letra);

Passagem de *strings* para funções

A passagem de *strings* para funções é idêntica à passagem de vetores para funções, dado que uma *string* é um vetor de caracteres.