

Licenciatura em Engenharia Informática – 2023/24

Programação

2. Ponteiros

Francisco Pereira (xico@isec.pt)

- Escrever uma função que troque o valor entre duas variáveis inteiras passadas como argumento:

```
int main() {  
    int x=1, y=2;  
  
    printf("Antes -> X: %d \t Y: %d\n", x, y);  
    troca(...);  
    printf("Depois -> X: %d \t Y: %d\n", x, y);  
    return 0;  
}
```

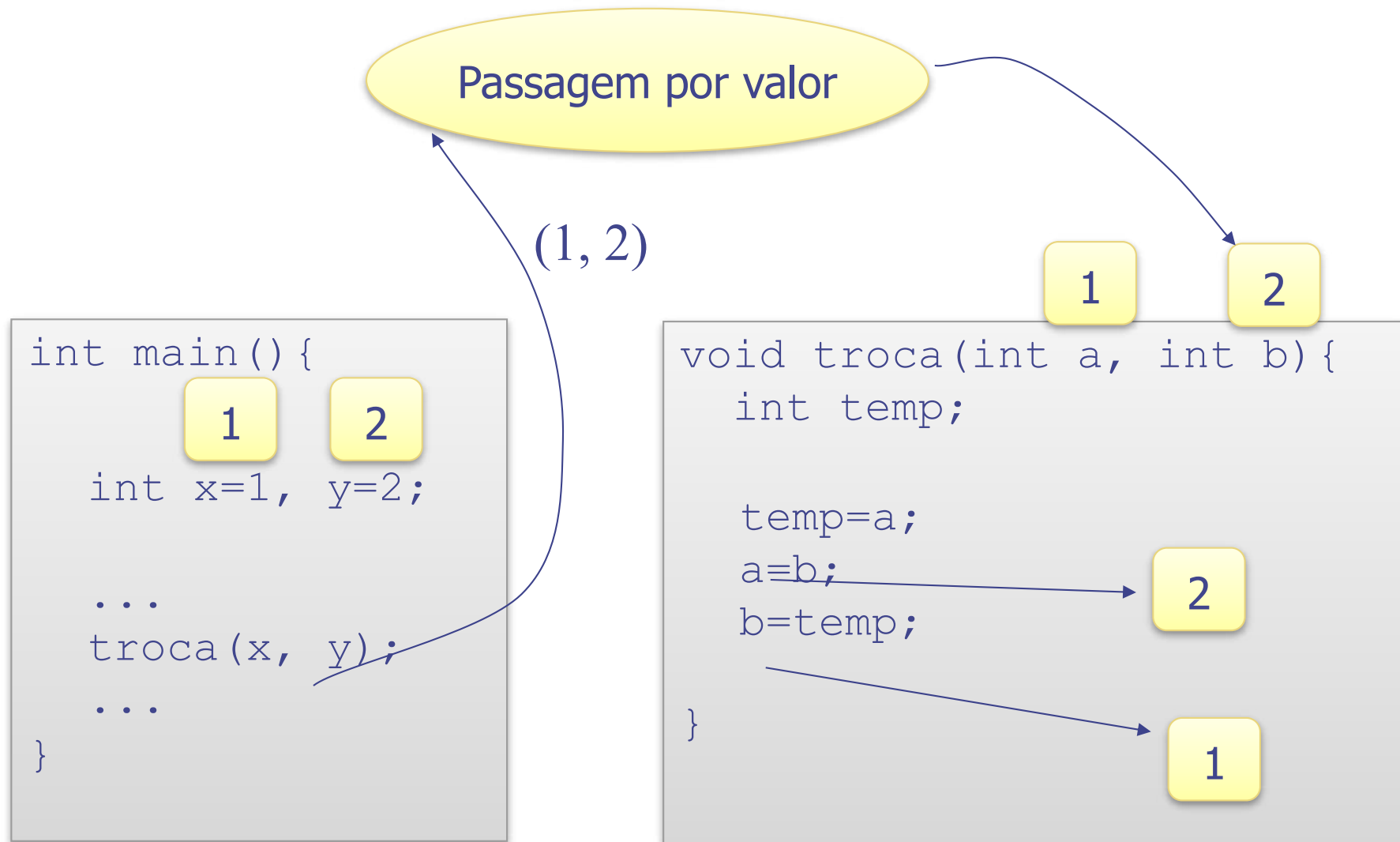
```
void troca(int a, int b){  
    int temp;  
  
    temp=a;  
    a=b;  
    b=temp;  
}
```

- Resultado:

Antes -> X: 1 Y: 2

Depois -> X: 1 Y: 2

Porquê?



Como resolver o problema?

- Utilizar ponteiros para estabelecer a comunicação entre funções

Pointers are a fundamental part of C. If you cannot use pointers properly then you have basically lost all the power and flexibility that C allows.

Organização da memória

- Declaração

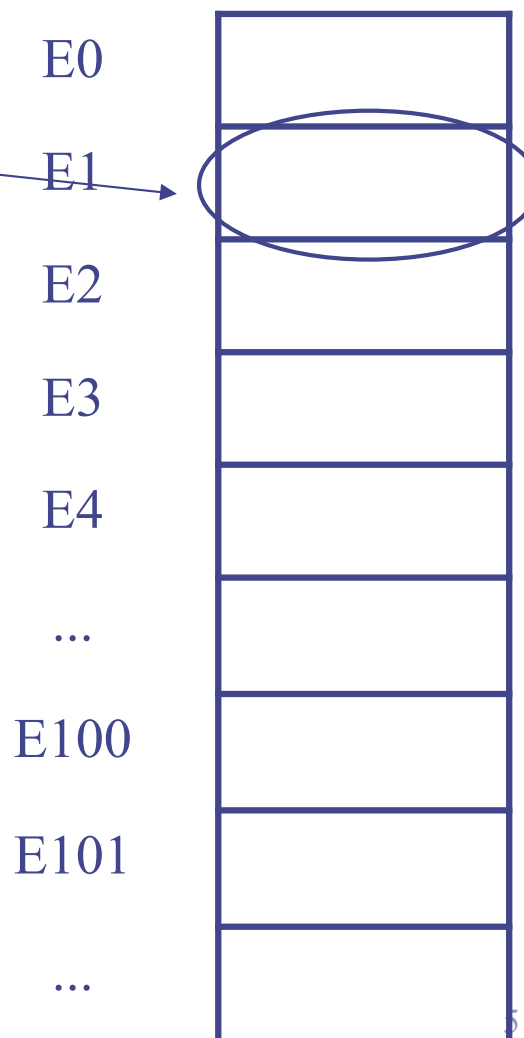
```
int a;
```

Vantagem: único

Endereço

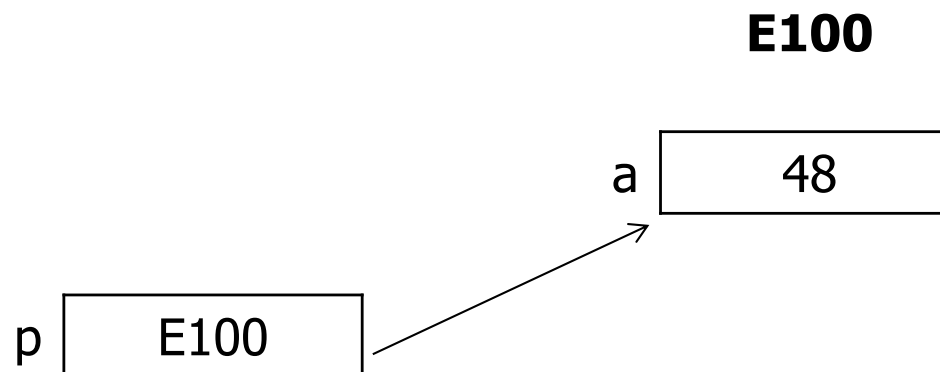
Conteúdo

- O identificador "a" fica associado a um espaço da memória
- Há duas maneiras para aceder a esse espaço.
 - Quais são?



Variáveis do tipo **Ponteiro**

- Guardam o endereço de outra variável
 - **Um ponteiro aponta para outra variável**
 - São usados como alternativa para aceder a uma variável



- Cada variável ponteiro é específica de um determinado tipo

- Declaração:

`tipo* nome_prt;`

```
int* p;  
int x;
```

- Dois operadores:

& : Obter o endereço de uma variável

***** : Aceder à variável para onde um ponteiro aponta

```
x = 10;
```

```
p = &x;
```

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

```
printf("%d", *p);
```

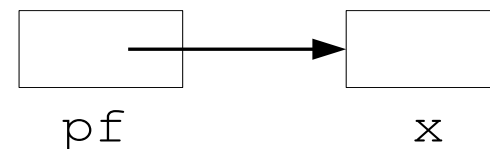

Declaração e operadores básicos

```
int a, b = 4;  
float x = 1.0;  
float* pf;  
int* p;
```

```
pf = &x;  
x = *pf + 2.5;  
printf("%f", x);  
p = &a;  
*p = b * 10;  
p = &b;  
printf("%d %d", *p, *p+a);
```

Ponteiros: 3 etapas

1. Declarar
2. Associar
3. Utilizar



**pf e x são expressões equivalentes*

Um ponteiro é uma variável

- Qual o resultado da execução?

Inicialização na
declaração

```
int main(){
    int i = 2, j = 3, *p = &i, *q = NULL;

    printf("Morada: %p\tValor: %d\n", p, *p);

    q = p;
    p = &j;
    *p = *q;
    (*q)++;
    printf("%d\t%d\n", i, j);
    return 0;
}
```

Atribuição
entre
ponteiros

- Os ponteiros resolvem o problema de comunicação entre as funções *troca()* e *main()*
 - Argumentos permitem acesso direto às variáveis a modificar
 - Versão 2 da função troca()

```
void troca_v2(int *p1, int *p2) {  
    int temp;  
  
    temp= *p1;  
    *p1=*p2;  
    *p2=temp;  
}
```

Função Troca: Versão Correta

Passagem dos endereços

```
int main()
{
    E25
    E47
    1  2
    int x=1, y=2;
    ...
    troca_v2 (&x, &y);
    ...
}
```

```
void troca_v2(int *p1, int *p2)
{
    int temp;

    temp= *p1;
    *p1=*p2;
    *p2=temp;
}
```

(E25, E47)

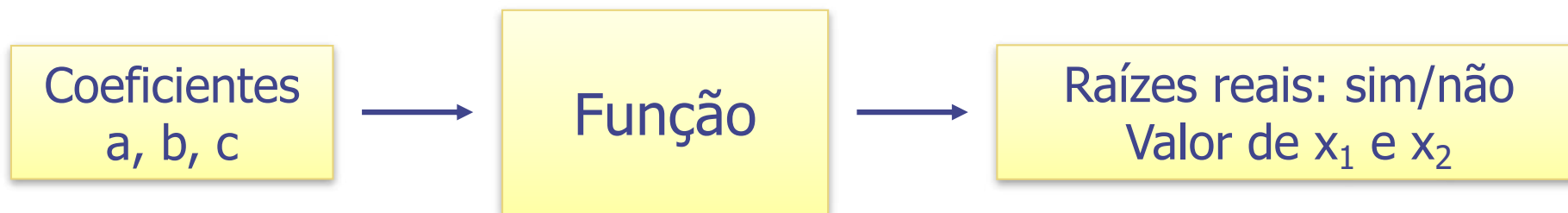
- Para alterar o valor de uma variável dentro de uma função:
 - Passar como argumento o endereço da variável e não uma cópia do seu valor.
 - Exemplo

```
int x;  
  
scanf ("%d", &x);
```

- Que erros existem neste código?

```
int main() {  
    int x = 10, *p, *q=&x;  
  
    *p = 100;  
  
    if( q == 10)  
        printf("DEZ!!");  
  
    *p = &x;  
  
    q = 100;  
    return 0;  
}
```

- Escrever uma função que obtenha as raízes reais de um polinómio do segundo grau
 - Exemplo: $x^2 - x - 6 = 0$
 - Raízes reais: $x_1 = -2; x_2 = 3$



```
#include <stdio.h>
#include <math.h>

int raizes(int a, int b, int c, double *x1, double *x2);

int main(){
    int a=1, b=-1, c=-6;
    double x1, x2;
    int reais;

    reais = raizes(a, b, c, &x1, &x2);
    if(reais == 1)
        printf("X1=%f, X2=%f\n", x1, x2);
    else
        printf("Raizes complexas :(\n");
    return 0;
}
```



```
int raizes(int a, int b, int c, double *x1, double *x2){  
    double delta;  
  
    delta = b*b - 4*a*c;  
    if (delta < 0)  
        return 0;  
  
    *x1 = (-b + sqrt(delta)) / (2*a);  
    *x2 = (-b - sqrt(delta)) / (2*a);  
    return 1;  
}
```

- Função para preencher um *array* de inteiros
 - Dois argumentos: o *array* e a sua dimensão

```
#define N 5

int main(){
    int i;
    int tab[N];

    inicializa(tab, N);

    for(i=0; i<N; i++)
        printf("%d\t", tab[i]);
    return 0;
}
```


Notação array

```
void inicializa(int a[], int tam) {  
    int i;  
  
    for(i=0; i<tam; i++)  
        a[i] = 2*i;  
}
```

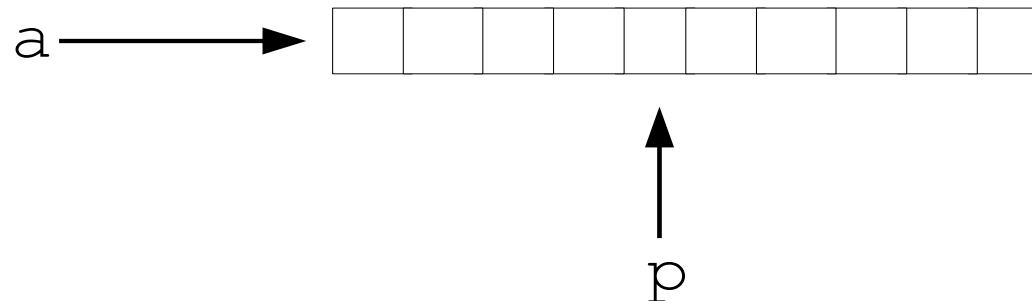
Resultado:

0	2	4	6	8
---	---	---	---	---

Porquê?

- Propriedade 1
 - O nome de um array é um ponteiro para o primeiro elemento.
- Propriedade 2
 - Os elementos do array estão em posições consecutivas de memória
- Num array unidimensional
 - `int a[10];` $a \longrightarrow$ 
 - `a[0]` ou `*a` representam o primeiro elemento.
 - O ponteiro `a` não pode ser modificado.

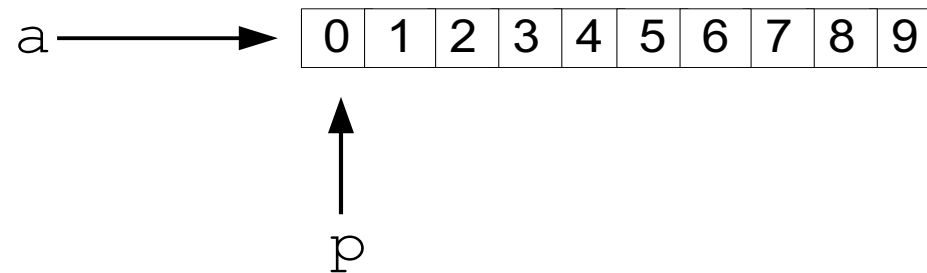
- Propriedade 3: Aritmética de Ponteiros
 - Se existir um ponteiro a referenciar um elemento de um array, é possível aceder aos restantes.
 - Exemplo:
 - Se o ponteiro p apontar para o elemento na posição X
 - $(p+i)$ aponta para o elemento na posição $(X+i)$
 - $(p-i)$ aponta para o elemento na posição $(X-i)$



Equivalência entre notações

- `int a[10] = {0,1,2,3,4,5,6,7,8,9};`

- `int *p = a;`



- As seguintes expressões são equivalentes:

<code>a</code>	é o mesmo que	<code>&a[0]</code>
<code>a[i]</code>	é o mesmo que	<code>*(a+i)</code>

- O programador deve ter cuidado com os limites!

Notação ponteiro

```
void inicializa_v2(int *a, int tam){  
    int i;  
  
    for(i=0; i<tam; i++)  
        *(a+i) = 2*i;  
}
```

Notação ponteiro

```
void inicializa_v3(int *a, int tam){  
    int i;  
  
    for(i=0; i<tam; i++)  
        *(a++) = 2*i;  
}
```


Encontre os Erros

```
int a=4, tab[3] = {10,20,30}, *p, *q;
```

```
p = &a;
```

```
q = tab + 2;
```

```
(*p)++;
```

```
p++;
```

```
*(tab + 1) = *p;
```

```
*p = tab;
```

```
tab[0] = q - tab;
```

```
tab[2] = (q > tab);
```

```
tab++;
```

Exercício

- Como descobrir se existem 2 elementos iguais num conjunto não ordenado?



- Pegar em cada um dos elementos e comparar com todos os outros



```
int repetidos(int *v, int tam);
```

- Recebe:

- Vetor v de inteiros com tam posições

- Devolve:

- 1 se existirem repetições
- 0 se todos os elementos forem únicos

```
int repetidos(int *v, int tam) {  
    int *p=v, *q, i;  
  
    for (i=0; i<tam- 1; i++) {  
        for (q = v + tam - 1; q != p; q--) {  
            if (*p == *q)  
                return 1;  
        }  
        p++;  
    }  
    return 0;  
};
```

- Descobrir os valores que surgem simultaneamente em 2 vetores de inteiros
 - Os valores em cada vetor são únicos
 - Os vetores podem ter tamanho diferente
 - Os vetores estão ordenados de forma crescente

4	5	8	12	15	16
----------	----------	----------	-----------	-----------	-----------

3	5	6	7	9	11	12	14
----------	----------	----------	----------	----------	-----------	-----------	-----------

```
int comuns(int *a, int tamA, int *b, int tamB);
```

- Recebe vetores
- Escreve elementos em comum na consola
- Devolve número de elementos comuns

- Exercício 1:
 - Implementar a função `escreve_valor`:
 - Recebe informação sobre uma matriz de inteiros já preenchida
 - Ponteiro para início, número de linhas e número de colunas
 - Escreve na consola o valor armazenado numa posição específica indicada pelo utilizador

- **Importante:** Programa lida com matrizes de diferentes dimensões

Matriz m1

1	2	3
4	5	6

Matriz m2

2	4
6	8

Matriz m3

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

Exercício 1

```
int main() {  
  
    int m1[2][3] = {{1,2,3},{7,8,9}};  
    int m2[2][2] = {{2,4},{6,8}};  
    int m3[3][4] = {{0,1,2,3},{4,5,6,7},{8,9,10,11}};  
  
    escreve_valor(m1, 2, 3);  
  
    escreve_valor(m2, 2, 2);  
  
    escreve_valor(m3, 3, 4);  
  
    ...  
}
```

Solução incorreta

```
void escreve_valor(int a[][], int n_lin, int n_col){  
    int x, y;  
  
    printf("Linha e coluna: ");  
    scanf("%d %d", &x, &y);  
  
    printf("%d\n", a[x][y]);  
}
```

- Erro de compilação!

```
void escreve_valor1(int a[][3], int n_lin, int n_col){  
    int x, y;  
  
    printf("Linha e coluna: ");  
    scanf("%d %d", &x, &y);  
  
    printf("%d\n", a[x][y]);  
}
```

- Desvantagem:
 - Só funciona corretamente com a matriz *m1[2][3]*
 - Falta de flexibilidade do código

Solução pouco prática (continuação)

```
void escreve_valor2(int a[][2], int n_lin, int n_col){  
  
    int x, y;  
  
    printf("Linha e coluna: ");  
    scanf("%d %d", &x, &y);  
  
    printf("%d\n", a[x][y]);  
}
```

```
void escreve_valor3(int a[][4], int n_lin, int n_col){  
  
    int x, y;  
  
    printf("Linha e coluna: ");  
    scanf("%d %d", &x, &y);  
  
    printf("%d\n", a[x][y]);  
}
```

Esta abordagem não
faz sentido

Solução Ideal (Abordagem Genérica)

Ordem dos argumentos

```
void escreve_valor(int n_lin, int n_col, int a[n_lin][n_col]) {  
  
    int x, y;  
  
    printf("Posição: ");  
    scanf("%d%d", &x, &y);  
    printf("Valor: %d\n", a[x][y]);  
}
```

■ Chamadas:

```
escreve_valor(2, 3, m1);  
escreve_valor(2, 2, m2);  
escreve_valor(3, 4, m3);
```

C99

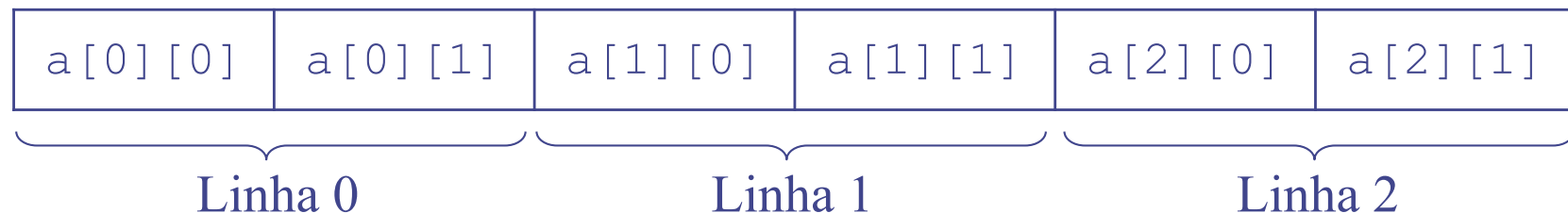
Arrays Bidimensionais: Organização em memória

```
#define L 3  
#define C 2  
  
int a[L][C];
```

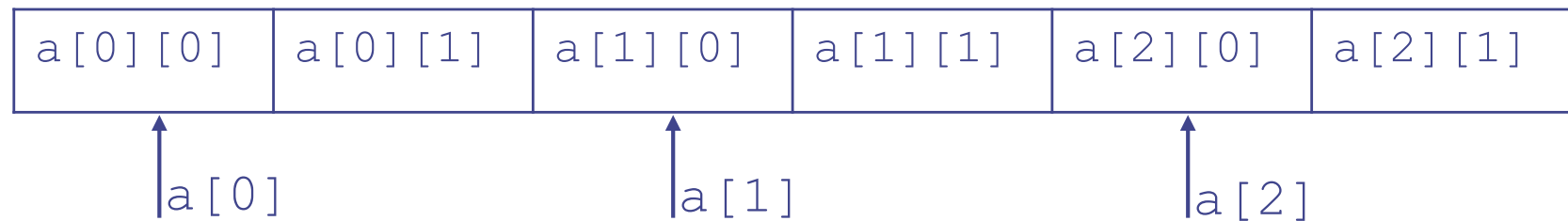
Organização
lógica

	0	1
0		
1		
2		

- Organização interna em memória:
 - Linhas armazenadas sequencialmente



- Num array bidimensional as linhas têm nome:
 - $a[i]$ é um ponteiro para o primeiro elemento da linha i



Exemplo: Somar diagonal principal

```
int main() {  
    int a[2][2] = {{2,4},{6,8}};  
  
    printf("%d\n", diag(2, a));  
  
    ...  
}
```

2	4
6	8

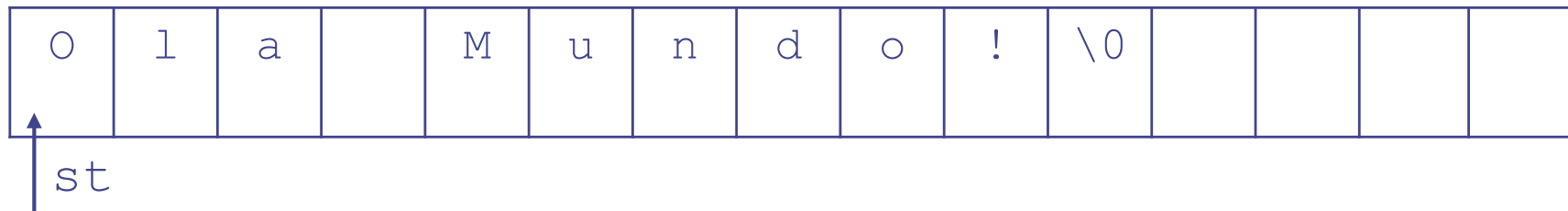
```
int diag(int n, int a[n][n]) {  
    int soma = 0, i;  
  
    for(i=0; i<n; i++)  
        soma += a[i][i];  
  
    return soma;  
}
```

```
int diag(int n, int a[n][n]) {  
    int soma = 0, i;  
  
    for(i=0; i<n; i++)  
        soma += *(a[i]+i);  
  
    return soma;  
}
```

char a[] VS. char *p

- Array de caracteres:

- `char st[15] = "Ola Mundo!";`



- Ponteiro para caracter:

- `char *p = "Ola Mundo!";`

Onde fica guardada
esta informação?

- Semelhanças:
 - `st` e `p` são dois ponteiros para caracter
- Diferenças:
 - Em relação à informação armazenada:
 - No array pode ser alterada
 - String referenciada por `p` deve ser considerada constante
 - Em relação aos ponteiros
 - `st` é constante
 - `p` é uma variável

- Armazenar um conjunto de frases que vai ser utilizado no programa:
 - "Out of range"
 - "Printer off-line"
 - "Paper out"
 - "Irrecoverable error"
- Que estrutura utilizar para o armazenamento?

Solução 1: Arrays de strings

```
char msg[][20] = {"Out of range",  
                  "Printer off-line",  
                  "Paper out",  
                  "Irrecoverable error"};
```

O	u	t		o	f		r	a	n	g	e	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
P	r	i	n	t	e	r		o	f	f	-	l	i	n	e	'\0'	'\0'	'\0'	'\0'
P	a	p	e	r		O	u	t	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'	'\0'
I	r	r	e	c	o	v	e	r	a	b	l	e		e	r	r	o	r	'\0'

- Escrever todas as mensagens de erro:

```
#define NUM_MSG 4

int main() {

    int i;
    char msg[][20]= ...; /*falta inicialização*/

    for(i=0; i<NUM_MSG; i++)
        puts(msg[i]);
    return 0;
}
```

- As mensagens, tendo tamanhos diferentes, conduzem a desperdício de espaço
 - Neste caso, 25% do espaço reservado não é aproveitado

Solução 2: Array de ponteiros p/ caracter

```
char *p_msg[] = { "Out of range",  
                  "Printer off-line",  
                  "Paper out",  
                  "Irrecoverable error"};
```

As frases não devem
ser alteradas!

- Escrever todas as mensagens de erro:

```
#define NUM_MSG 4

int main() {

    int i;
    char *p_msg[] = ...; /*falta inicialização*/

    for(i=0; i<NUM_MSG; i++)
        puts(p_msg[i]);
    return 0;
}
```


- Acesso a um caracter específico:
 - Escrever as mensagens que se iniciam com 'P'
 - Versão array de strings:

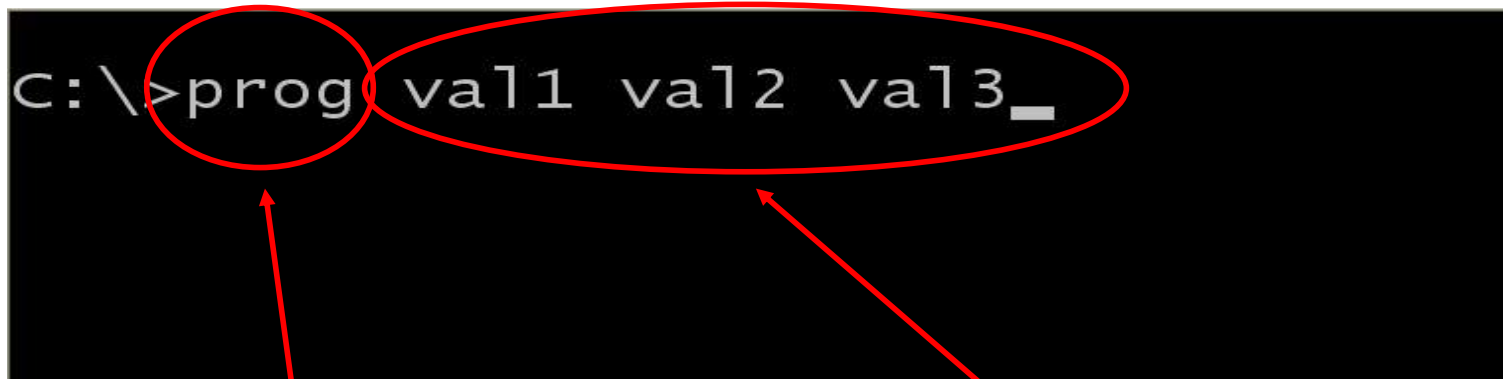
```
for(i=0; i<NUM_MSG; i++)  
    if(msg[i][0] == 'P')  
        puts(msg[i]);
```

- Versão array de ponteiros para caracter:

```
for(i=0; i<NUM_MSG; i++)  
    if(p_msg[i][0] == 'P')  
        puts(p_msg[i]);
```

Argumentos de um programa

- Como transmitir informação a um programa quando este inicia a sua execução?
 - Argumentos passados à função `main()`
 - Argumentos do programa (ou da linha de comando)



A screenshot of a Windows command prompt window with a black background and white text. The command line shows 'c:\>prog val1 val2 val3_'. A red circle highlights the word 'prog', and another red circle highlights the space-separated values 'val1 val2 val3_'. Two red arrows point from yellow boxes below to these circles: one from the 'Nome do programa' box to the 'prog' circle, and one from the 'Argumentos' box to the 'val1 val2 val3_' circle.

```
c:\>prog val1 val2 val3_
```

Nome do
programa

Argumentos

- Esqueleto da função `main()`:

```
int main(int argc, char *argv[]){  
  
    ...  
  
    return 0;  
}
```

- `argc`
 - Número de argumentos na linha de comando
 - Inclui o nome do programa
- `argv`
 - Cada elemento do array aponta para um argumento da linha de comando
 - Elementos são considerados strings
 - Tem $(argc+1)$ elementos (o último tem o valor `NULL`).

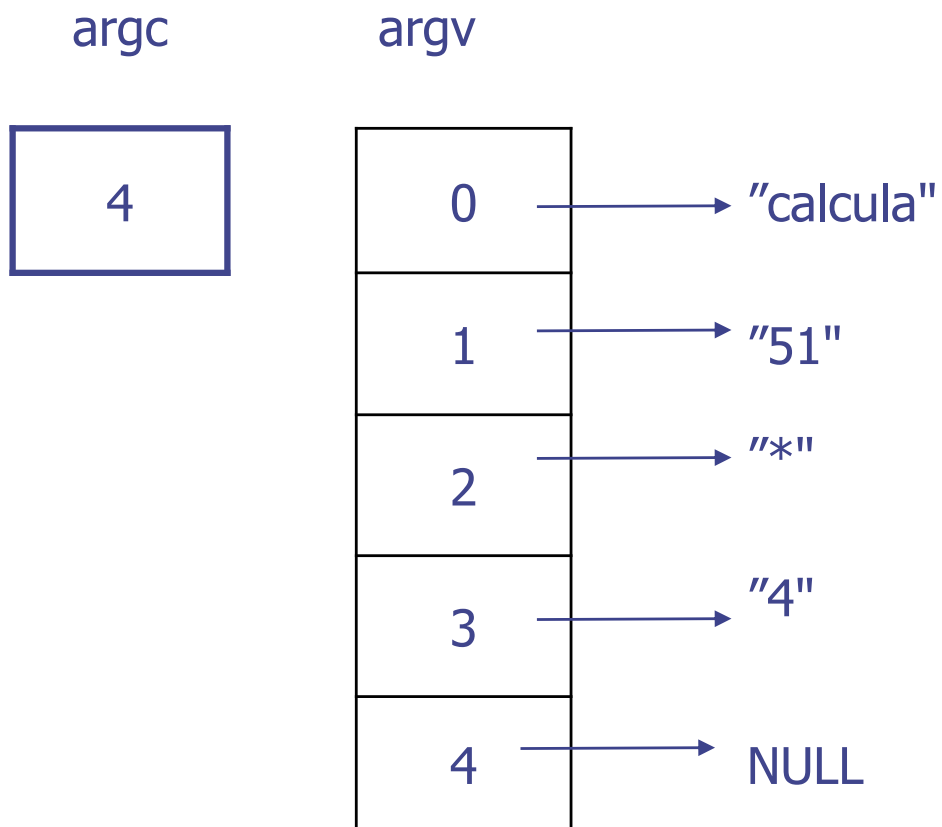
- Implementar um programa chamado `calcula` que efetue as operações aritméticas básicas
 - Recebe dois inteiros e uma operação (+, -, *, /)
 - Escreve o resultado no monitor

```
C:\> calcula 51 + 43
```

```
Total = 94
```

Exercício

```
C:\> calcula 51 + 43
```



Solução

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
    int x, y;
    if(argc != 4) printf("Utilizacao incorrecta\n");
    else{
        x = atoi(argv[1]);
        y = atoi(argv[3]);
        switch(*argv[2]){
            case '+': printf("Total: %d\n", x+y); break;
            case '-': printf("Total: %d\n", x-y); break;
            case '*': printf("Total: %d\n", x*y); break;
            case '/': if(y!=0)
                        printf("Total = %d\n", x/y); break;
            default: printf("Operacao invalida\n");
        }
    }
    return 0;
}
```