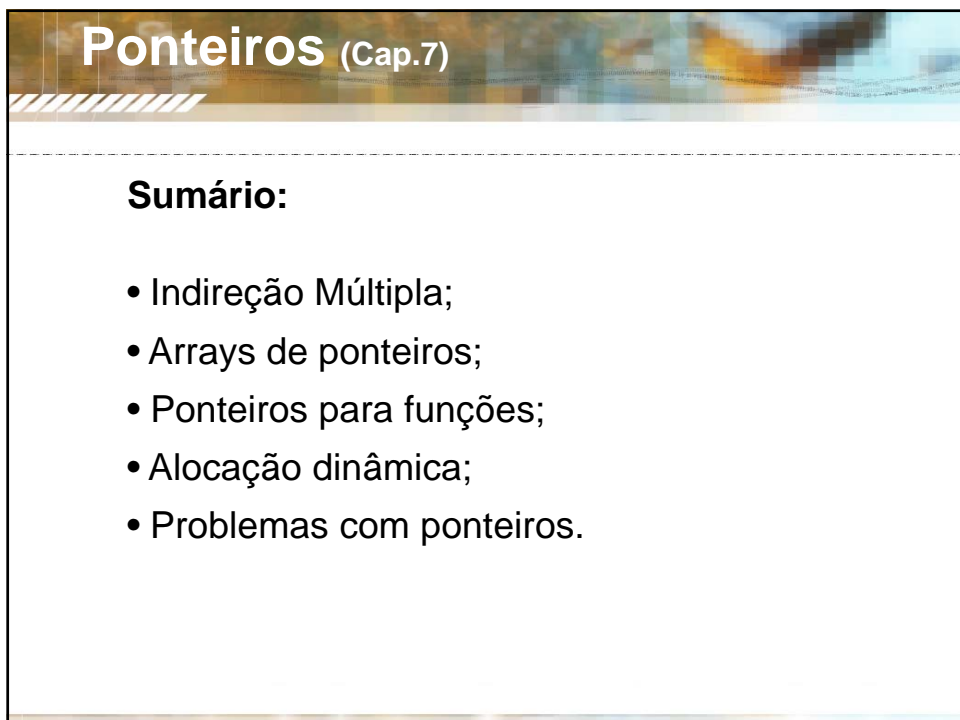


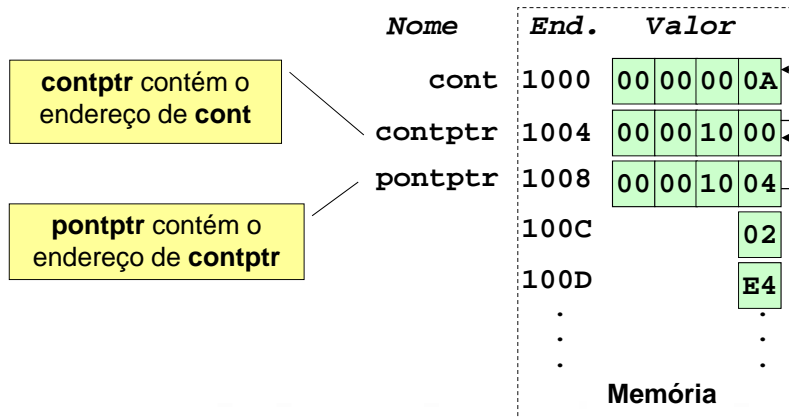
Ponteiros aula 2

Capítulo 7



Indireção Múltipla

Pode-se ter um ponteiro apontando para outro ponteiro que aponta para o valor final. Essa situação é chamada **indireção múltipla**, ou **ponteiros para ponteiros**.



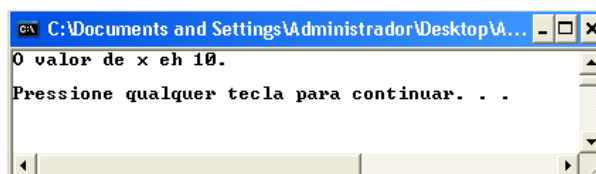
Indireção Múltipla

Exemplo:

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

main()
{
    int x, *p, **q;

    system("color f0");
    x=10;
    p=&x;
    q=&p;
    printf("O valor de x eh %d.\n\n", **q);
    system("pause");
}
```



Arrays de Ponteiros (Cap.7)

Arrays podem conter ponteiros.

- A declaração para um array contendo 10 ponteiros é feita da seguinte forma:

```
int *x[10];
```

- O terceiro ponteiro do array x passa a conter o endereço de var:

```
x[2] = &var;
```

- O valor apontado pelo terceiro ponteiro do array x é retornado:

```
*x[2]
```

Arrays de Ponteiros (Cap.7)

Ex:

```
char *naipe[4] = {"copas", "ouros", "paus", "espadas"}
```

“naipe” é um array de quatro posições cujos elementos (são endereços) que apontam para caracteres.

Cada uma das cadeias de caracteres são armazenados em um dado local disponível de memória e são terminadas por “\0”. Cada um dos ponteiros do *array* aponta para o endereço do primeiro caractere de cada uma delas.

Vantagem: Não é necessário alocar um espaço “retangular”.

Arrays de Ponteiros (Cap.7)

Exemplo:

```
main()  
{  
    char *naipe[4]={ "copas", "ouros", "paus", "espadas"};  
    int t;  
  
    system("color f0");  
    printf("As 4 String sao:\n");  
    for (t=0;t<4;t++)  
        printf("Naipe [%d] --> %7s\n",t,naipe[t]);  
    system("pause");  
}
```

```
As 4 String sao:
Naipes [0] --> copas
Naipes [1] --> ouros
Naipes [2] --> paus
Naipes [3] --> espadas
Pressione qualquer tecla para continuar. . .
```

Arrays de Ponteiros (Cap.7)

```
char *naipes[4] = {"copas", "ouros", "paus", "espadas"}
```

As strings **não** são colocadas em **naipe**

<i>Nome</i>	<i>End.</i>	<i>Valor</i>		<i>End.</i>	<i>Valor</i>
naipe	4020	00 00 80 00	→	8000	c o p a s \0
	4024	00 00 80 06	→	8006	o u r o s \0
	4028	00 00 80 0C	→	800C	p a u s \0
	402C	00 00 80 11	→	8011	e s p a d a s \0
	402D	3B		8019	65

.	.		.	.	
.	.		.	.	
	Memória				

São inseridos em **naipe** ponteiros para as strings

Memória

Ponteiros para funções

Uma função também possui uma localização física na memória. Este endereço é o início da função na memória e é passado ao programa quando a função é chamada.

Logo, este endereço pode ser atribuído a um ponteiro que pode ser passado a outra função como variável.

Declaração:

tipo (*nome) (argumentos)

Aqui serão declaradas as variáveis que são passadas à função à qual o ponteiro faz referência

ANALOGIA:

Nome do Array → endereço inicial onde está armazenado;

Nome da função → endereço inicial (onde está armazenado) do código que realiza a tarefa.

Ponteiros para funções

```
void check(char *a, char *b, int (*cmp)(const char*, const char*));
```

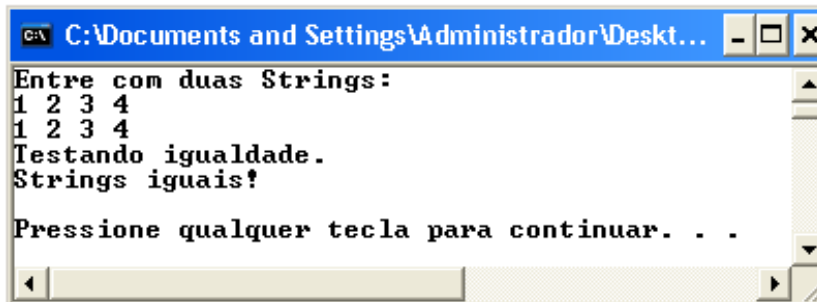
```
main()
{
    char s1[80], s2[80];
    int (*p)(const char*, const char*);
    system("color f0");
    p=strcmp;
    printf("Entre com duas Strings:\n");
    gets(s1);
    gets(s2);
    check(s1, s2, p);
    system("pause");
}
```

strcmp compara duas strings, seu endereço é guardado em p

```
void check(char *a, char *b, int (*cmp)(const char*, const char*))
{
    printf("Testando igualdade.\n");
    if ((*cmp)(a, b) == 0)
        printf("Strings iguais!\n");
    else
        printf("Strings diferentes!\n\n");
}
```

Chama a função strcmp

Ponteiros para funções



```
C:\Documents and Settings\Administrador\Desktop>
Entre com duas Strings:
1 2 3 4
1 2 3 4
Testando igualdade.
Strings iguais!

Pressione qualquer tecla para continuar. . .
```

Alocação Dinâmica

Alocação dinâmica é o meio pelo qual um programa pode obter memória enquanto está em execução. A função **malloc()** aloca memória e a função **free()** libera a memória anteriormente alocada.

Protótipos:

malloc retorna um ponteiro para o primeiro endereço de memória livre

```
void *malloc(size_t numero_de_bytes);
```

```
void free(void *p);
```

Espaço que se deseja reservar

Libera o espaço apontado por p

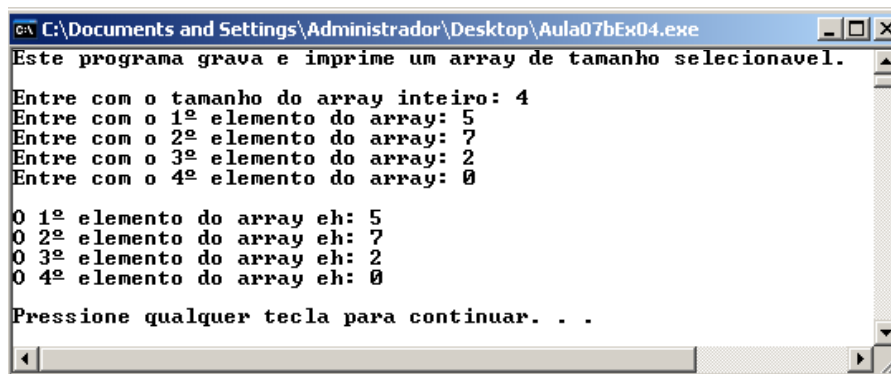
Alocação Dinâmica

```
main()
{
    int *ptr, tamanho, i;
    printf("Este programa grava e imprime um array de "
           "tamanho selecionavel.\n\n");
    printf("Entre com o tamanho do array inteiro: ");
    scanf("%d",&tamanho);
    ptr=malloc(tamanho*sizeof(int));
    if(ptr==0())
        printf("A requisição de memória falhou!\n");
    else
    {
        for(i=0; i<tamanho; i++)
        {
            printf("Entre com o %d%c elemento do array: ", i+1, 167);
            scanf("%d", ptr+i);
        }
        for(i=0; i<tamanho; i++)
            printf("\n0 %d%c elemento do array eh: %d", i+1, 167, ptr[i]);
    }
    printf("\n\n");
    system("pause");
}
```

Grava em **ptr** o endereço

Se **malloc()** não encontrar espaço na memória, **ptr** é igual a 0, atendendo a condição.

Alocação Dinâmica

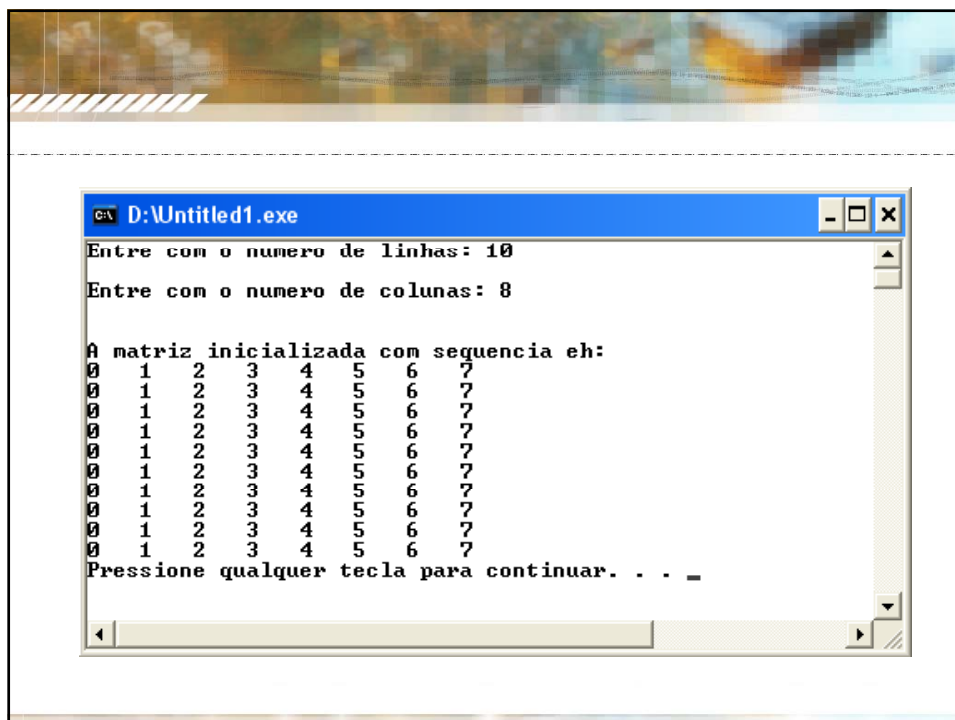
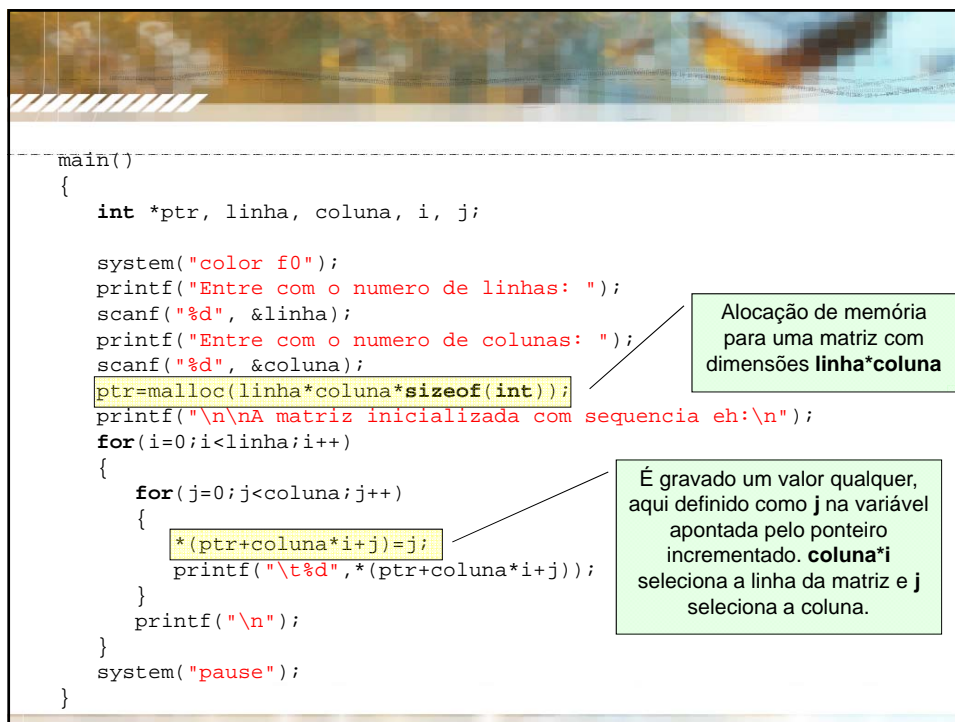


```
C:\Documents and Settings\Administrador\Desktop\Aula07bEx04.exe
Este programa grava e imprime um array de tamanho selecionavel.

Entre com o tamanho do array inteiro: 4
Entre com o 1º elemento do array: 5
Entre com o 2º elemento do array: 7
Entre com o 3º elemento do array: 2
Entre com o 4º elemento do array: 0

0 1º elemento do array eh: 5
0 2º elemento do array eh: 7
0 3º elemento do array eh: 2
0 4º elemento do array eh: 0

Pressione qualquer tecla para continuar. . .
```

Problemas com ponteiros

Quando se comete algum erro na utilização de ponteiros corre-se um grande risco. Este pode alterar o valor contido em um endereço que está sendo usado por outros programas ou então pelo sistema operacional, e pode causar erros ou até travar o computador.

Problemas com ponteiros

Exemplos

//Este programa está errado!

```
main()  
{  
    int x, *p;  
    x=10;  
    *p=x;  
    printf("%d", *p);  
}
```

Aqui é atribuído o valor de **x** ao endereço apontado por **p**. Mas este endereço não foi inicializado. O ponteiro aponta para um lugar desconhecido (pode ser um endereço não seguro).

Problemas com ponteiros

Exemplos

//Este programa está errado!

```
main()  
{  
    int x, *p;  
    x=10;  
    p=x;  
    printf("%d", *p);  
}
```

O ponteiro **p** passa a apontar para o endereço "10", seu conteúdo é diferente de **x**. Para atribuir o endereço de **x** a **p**, deveria ser trocada por **p = &x**;

Problemas com ponteiros

Exemplos

//Este programa está errado!

```
main()  
{  
    char s[80], y[80];  
    char *p1, *p2;  
    p1 = s;  
    p2 = y;  
    if(p1 < p2)  
        ...  
}
```

O programa pode ser executado em diferentes máquinas e em cada uma delas **s** e **y** podem ser alocados em endereços distintos de memória. Portanto uma comparação entre os endereços para uma condição de funcionamento faz com que o programa possa não se comportar da mesma forma em diferentes máquinas.