

Linguagem de Programação

Estrutura de um Programa

- Ponteiros
- Material: LP_Aula08

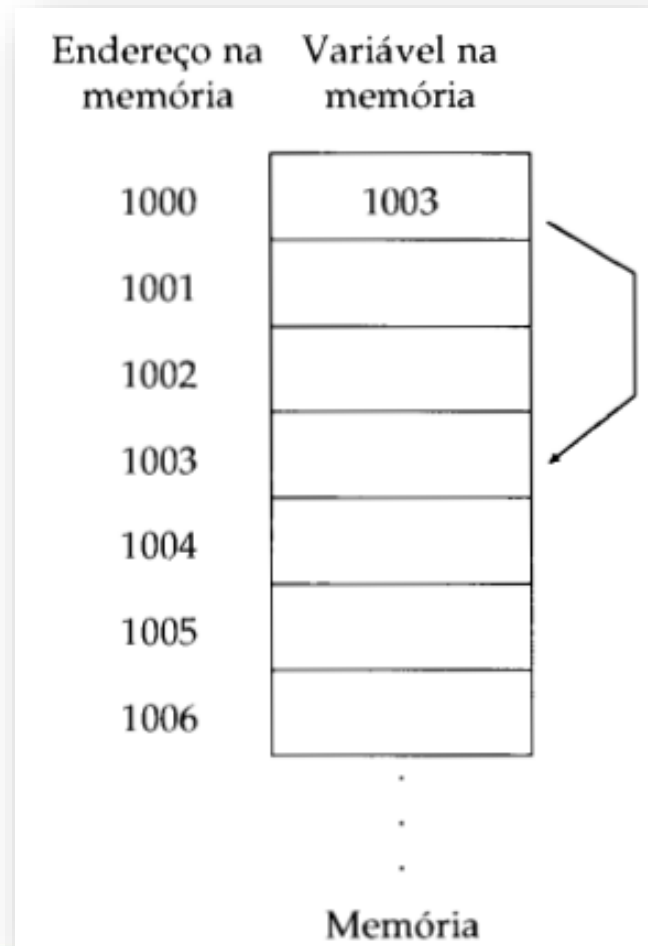


Diz Herbert(1997)

- Um ponteiro é uma variável com contém um endereço de memória.
- Normalmente esse endereço contém a posição de uma outra variável na memória.
- Se uma variável contém o endereço de uma outra, então a primeira variável é dita para *apontar* para a segunda.



Exemplo



Variáveis ponteiros

Para declarar uma variável ponteiro, deve-se colocar no tipo base um **'*'** (asterisco) e o nome da variável.

Sintaxe:

tipo *nome

Onde:

- o tipo pode ser **qualquer tipo válido em C;**
- e nome é o **nome da variável ponteiro.**



Operadores Ponteiros

Operadores especiais **&** e *****;

- O **&** é o operador que devolve o endereço na memória do seu operando.
- O operador ***** devolve o valor da variável localizada no endereço que o segue. Ou seja:

& = endereço

e

*** = conteúdo**



Carga Inicial dos Ponteiros

Faz-se através do operador & (Endereço de).

Exemplo:

```
int x = 5;  
float pi = 3.14;  
int * ptr_x = &x;  
float * pointer_to_pi = &pi;
```

Nota:

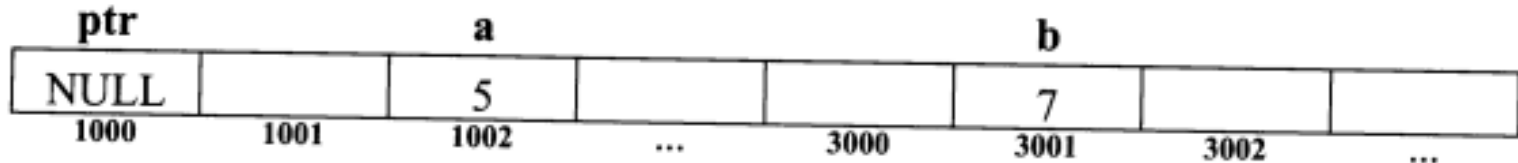
Um bom hábito para evitar problemas de programação é sempre a carga inicial dos ponteiros.



Usando Ponteiros

Um ponteiro serve para acessar outros objetos através dos seus endereços.

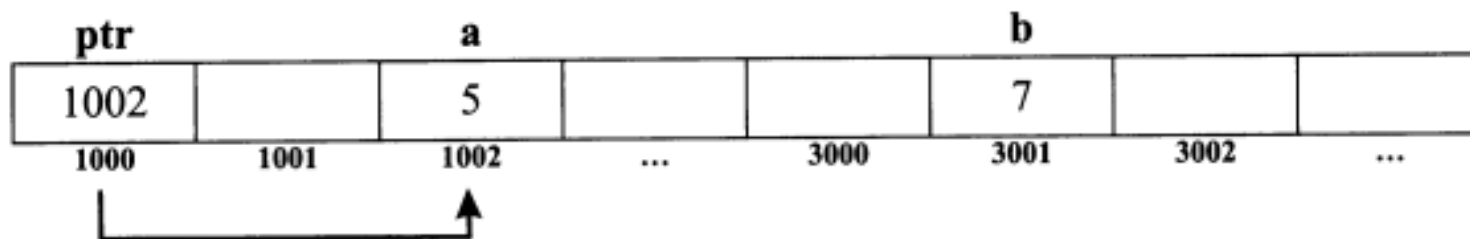
```
int a=5, b=7;  
int * ptr = NULL;           /* ptr não aponta para nada */
```



- Para se colocar **ptr** apontando para **a** faz-se:

```
ptr = &a;
```





Após a instrução anterior o valor de

a → 5

ptr → 1002

*ptr

→ 5

/* Endereço de *a* */

/* Aquilo que está na casa nº 1002 */

Nota: Se *ptr* é um ponteiro, então **ptr* nos permite obter o valor que é apontado por *ptr*, isto é, o valor da variável cujo endereço está armazenado em *ptr*.

***ptr** – Deve-se ler “O APONTADO POR *ptr*”.

Dessa forma, fazer

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

é equivalente a

```
printf("%d", *ptr)
```



```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int a = 12;
    int b = 0;
    int *pa = NULL;
    pa = &a;
    printf("Valor de pa: %d\n", *pa);
    a = 15;
    b = *pa;
    printf("Valor de b: %d\n", b);
    printf("Valor de a, b, pa e * pa --> %d %d %d %d\n", a,b,pa,*pa);
    system("PAUSE");
    return 0;
}
```

1000	1001	1002
a	b	pa
12	0	NULL
12	0	1000
15	15	1000

Ponteiros e Tipos de Dados

Ponteiros devem sempre possuir um tipo (e não ser genérico) devido ao fato de as variáveis ocuparem diferentes tamanhos em memória que os ponteiros para essas variáveis terão que saber quantos **bytes** terão que considerar.

```
char *ptr_a = &a;  
int *ptr_n = &n;  
float *ptr_pi = &pi;
```

A declaração anterior nos diz que o número de *bytes* que irá ser considerado por cada um dos ponteiros é 1, 2 e 4, respectivamente, pois cada ponteiro terá que considerar o número de *bytes* do tipo que aponta.



Operações aritméticas válidas com ponteiros

Operação	Exemplo	Observações
Atribuição	<code>ptr = &x</code>	Podemos atribuir um valor (endereço) a um ponteiro. Se quisermos que aponte para nada podemos atribuir-lhe o valor da constante NULL.
Incremento	<code>ptr=ptr+2</code>	Incremento de $2 * \text{sizeof}(\text{tipo})$ de ptr.
Decremento	<code>ptr=ptr-10</code>	Decremento de $10 * \text{sizeof}(\text{tipo})$ de ptr.
Apontado por	<code>*ptr</code>	O operador asterisco permite obter o valor existente na posição cujo endereço está armazenado em ptr.
Endereço de	<code>&ptr</code>	Tal como qualquer outra variável, um ponteiro ocupa espaço em memória. Dessa forma podemos saber qual o endereço que um ponteiro ocupa em memória.
Diferença	<code>ptr1 - ptr2</code>	Permite-nos saber qual o nº de elementos entre ptr1 e ptr2.
Comparação	<code>ptr1 > ptr2</code>	Permite-nos verificar, por exemplo, qual a ordem de dois elementos num vetor através do valor dos seus endereços.

Exemplo: (incremento)

Um ponteiro pode ser incrementado como qualquer variável.

Entretanto, o incremento de uma unidade não significa que o endereço anteriormente armazenado no ponteiro seja incrementado em um *byte*.



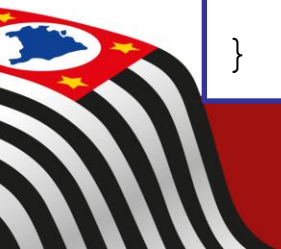
Exemplo (ponteiroinc.c)

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

int main(int argc, char *argv[])
{
    short x = 5, *px = &x;
    long y = 5.0, *py = &y;
    printf("%d %ld\n", x, (long)px);
    printf("%d %ld\n", x + 1, (long)(px + 1));

    printf("%d %ld\n", y, (long)py);
    printf("%d %ld\n", y + 1, (long)(py + 1));

    system("PAUSE");
    return 0;
}
```



Exemplo (decremento)

O decremento funciona da mesma forma que o incremento, ou seja, um ponteiro para o tipo xxx recua sempre o sizeof(xxx) bytes por unidade de decremento. O próximo exemplo mostra uma string na tela pela ordem em que está escrita e na ordem contrária.




```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    char s[100];
    char *ptr = &s; /* aponta para o 1o. caractere de s*/
    printf("Introduza uma string: ");
    gets(s);
    if (*ptr == '\0') return;
    /*Imprimir string normalmente */
    while(*ptr!='\0')
        putchar(*ptr++);
    printf("\n\n");
    /*Imprimir ao contrário */
    ptr--; //por causa do '\0'

    while(ptr>=s) /*Enquanto ptr for >=que &s[0] */
        putchar(*ptr--);
    printf("\n\n");
    system("PAUSE");
    return 0;
}
```

Exemplo (diferença)

A diferença entre dois ponteiros para elementos do mesmo tipo **permite saber quantos elementos existem entre um endereço e outro.**

Nota: A diferença entre ponteiros só pode ser realizada **entre ponteiros do mesmo tipo.**



```
#include <stdio.h>
#include <stdlib.h>
int strlen(char *s); /* Protótipo da função */

int main(int argc, char *argv[])
{
    char s[100];
    char *ptr = s; /* Aponta para o primeiro caractere de S */

    printf("Digite uma string: " ); gets(s);

    printf("%d\n", strlen(s));

    system("PAUSE");
    return 0;
}

int strlen(char *s)
{
    char *ptr=s; /* Guarda o endereço inicial */
    while(*s!='\0') /* Enquanto nao chegarmos ao fim */
        s++;
    return(int) (s - ptr); /* Retorna o a diferença entre os
                             endereços */
}
```



Exemplo (comparação)

É possível comparar dois ponteiros de um mesmo tipo, utilizando os operadores relacionais (<, <=, >, >=, == e !=).

Ex:

```
while(ptr >= s) /*Enquanto ptr for >= que &s[0] */  
    putchar(*ptr--);
```

A diferença e a comparação entre ponteiros só podem ser realizadas entre ponteiros de um mesmo tipo.



Sendo o nome de um vetor o endereço do seu primeiro elemento, poderemos realizar (com ele) todas as operações a que temos acesso quando manipulamos ponteiros, desde que essas operações não **alterem o seu valor**, pois o nome de um vetor é uma constante.



Exemplos

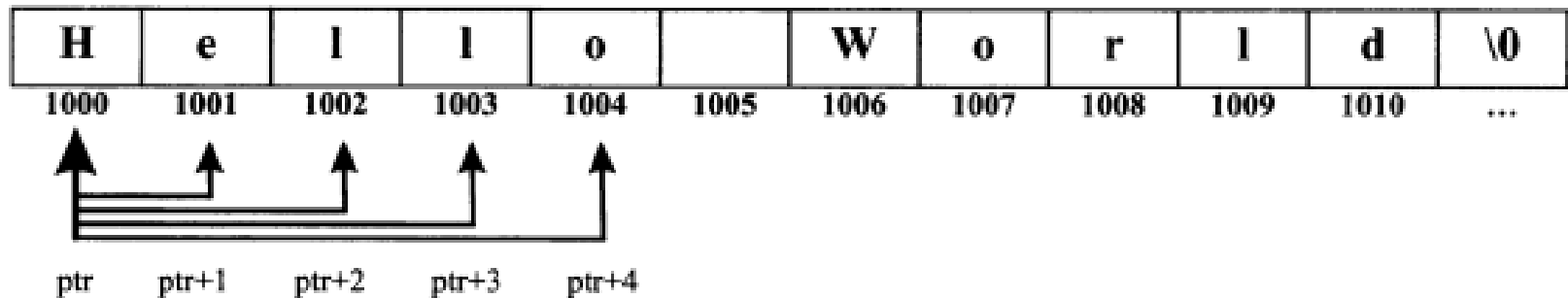
```
char s[20] = "Ola";
```

```
s="ole"      /* Erro de Compilação. Devia usar strcpy */
s++         /* Erro: Não podemos alterar s */
s+1         /* OK: Não estamos alterando s */
*s          /* OK. */
(*s)++      /* OK: Não estamos alterando s, mas sim um dos seus chars */
s = s-2     /* Erro: Não podemos alterar s */
s > s+1     /* OK. */
s+1-s       /* OK. */
```



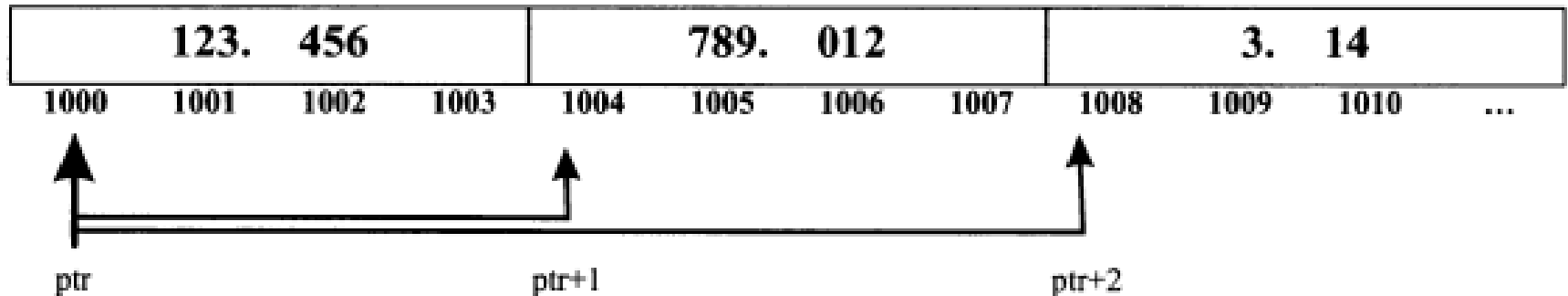
Exemplo (a)

Se ptr for um ponteiro para um vetor de caracteres, então



Exemplo (b)

Se *ptr* for um ponteiro para um vetor de *floats* (quatro *bytes* cada), então



Pergunta:

Explique as figuras anteriores.

O que se pode verificar em comum entre elas. Figura do Exemplo (a) e do Exemplo (b).



CENTRO PAULA SOUZA

Fatec

Mogi Mirim

Arthur de Azevedo

Ponteiros e vetores



Considere o exemplo:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int strlen(char *s)
4 {
5     char *ptr = s; //posicao inicial
6     while (*s != '\0')
7         s++; //posicao final
8     return (int) (s - ptr);
9 }
10 int main(int argc, char *argv[])
11 {
12     char nome[100];
13     printf("Digite um nome: "); gets(nome);
14     printf("%d\n", strlen(nome));
15     system("PAUSE");
16     return 0;
17 }
```



Exemplo:

Repare como o exemplo funciona: declara-se uma *string* denominada *Nome*, e suponhamos que o usuário introduza o nome Maria.

M	a	r	i	a	\0	
1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	...

Ao invocarmos a função `strlen` passamos como parâmetro a string `nome`.

A variável ponteiro `*ptr` guarda o endereço inicial de `nome` [endereço 1000].

O bloco `while` varre o vetor de caracteres até encontrar o “\0” incrementando a posição (endereço) em 1. [s no final está em 1005].
Retorno = $1005 - 1000 = 5$ (tamanho dos caracteres).



Nota do Maromo

Os slides desta **aula tomam por base o capítulo 8 do Livro do Damas** indicado nesta bibliografia.

Recomenda-se aos alunos efetuarem todos os **exercícios propostos neste capítulo** e dirimirem suas dúvidas com o professor.



Ponteiros e Vetores – Acesso aos Elementos

Suponha-se a seguinte declaração:

```
char palavra[] = "MacaMataMalaMapa";  
char *ptr = palavra; /* ptr fica com o endereço &palavra[0] */
```

De que formas se pode acessar o caractere “t” presente na string ?



Ponteiro de Ponteiros

Pergunta:

- Se você estiver interessado em armazenar o endereço de um ponteiro, qual o tipo de variável que irá recebê-lo?

Resposta:

[...] Variável comum: `int x;`

[...] Ponteiro para `x`: `int * ptr_x;`

[...] Ponteiro para o ponteiro de `x`: `int ** ptr_ptr_x`

E assim sucessivamente

Considere o exemplo:

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

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

```
int main(int argc, char *argv[])
```

```
{
```

```
    int x = 5;
```

```
    int * ptr_x;      //ponteiro para x
```

```
    int ** ptr_ptr_x; //ponteiro para ponteiro de x
```

```
    //Carga inicial dos ponteiros;
```

```
    ptr_x = &x;
```

```
    ptr_ptr_x = &ptr_x;
```

```
    printf("x = %d - Endereco de x = %ld\n", x, &x);
```

```
    printf("x = %d - Endereco de x = %ld\n", *ptr_x, ptr_x);
```

```
    printf("x= %d - Endereco de x = %ld\n", **ptr_ptr_x, *ptr_ptr_x);
```

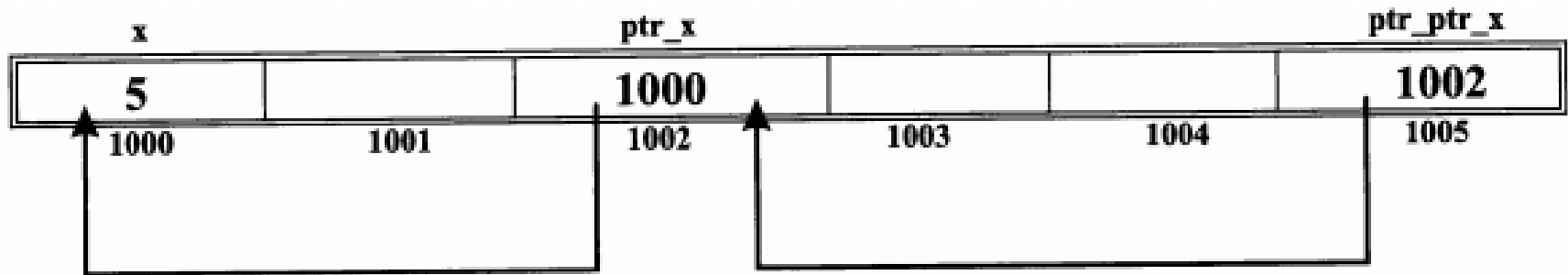
```
    system("PAUSE");
```

```
    return 0;
```

```
}
```



Supondo os endereços



Como se pode ver, **ptr_x** aponta para a variável **x**, enquanto **ptr_ptr_x** aponta para **ptr_x**

Expressão	Tipo	Valor	Descrição
x	int	5	
ptr_x	int *	1000	
*ptr_x	int	5	Valor Apontado por ptr_x
ptr_ptr_x	int **	1002	
*ptr_ptr_x	int *	1000	Valor Apontado por ptr_ptr_x
**ptr_ptr_x	int	5	Valor Apontado pelo endereço Apontado por ptr_ptr_x

Notas do Autor Damas

É necessário prestar atenção em alguns pontos sobre ponteiros:

1. Um ponteiro é uma variável que não tem memória própria associada (apenas possui o espaço para conter um endereço), apontando normalmente para outros objetos já existentes. Funciona, mais ou menos, como um comando de televisão.
2. Embora seja possível utilizá-los como vetores, os ponteiros não possuem memória própria. Só se pode utilizar o endereçamento através de um ponteiro depois que este está apontando para algum objeto já existente.
3. Não se deve fazer cargas iniciais de objetos apontados por um ponteiro que ainda não tenha sido iniciado.




Exercícios

1) Suponha a seguinte declaração:

```
int x=2, *px, *py, y=3;
```

que corresponde ao seguinte esquema de memória



		106
		105
		104
y	3	103
py		102
px		101
x	2	100

Supondo que a escrita de inteiros e ponteiros pode ser feita através da função printf usando o formato %d.

- Escreva o código necessário para colocar px apontando para x e py apontando para y.
- Depois de executado o item (a), qual a saída das seguintes instruções?

```
printf("%d %d\n", x, y);  
printf("%d %d\n", *px, *py);  
printf("%d %d\n", &px, &py);
```

- Caso seja feito px = py, qual a saída de:

```
printf("%d %d %d %d %d %d %d %d\n", x, &x, px,  
*px, y, &y, py, *py);
```



2) Suponha a seguinte declaração:

```
int v=7;
```

```
int *pv;
```

```
int **ppv;
```

- a) Qual o conjunto de instruções que você escreveria, de tal modo que as três variáveis ficassem de alguma forma, relacionadas entre si?
- b) Represente, em um esquema de memória, qual o estado em que ficou a memória depois de executadas as instruções do item (a), supondo que a dimensão da memória é de um Byte. Dê valores que ache apropriados aos endereços de todas as variáveis envolvidas.



CENTRO PAULA SOUZA

Fatec

Mogi Mirim

Arthur de Azevedo

Prof. Me. Marcos Roberto de Moraes, o Maromo

FIM



Referências Bibliográficas

DAMAS, L. M. D. Linguagem C. LTC, 2007.

HERBERT, S. C completo e total. 3a. ed. Pearson, 1997.

