SSC0101 - ICC1 – Teórica

Introdução à Ciência da Computação I

Estruturas Dinâmicas - Ponteiros
Parte I

Prof. Vanderlei Bonato: vbonato@icmc.usp.br
Prof. Claudio Fabiano Motta Toledo: claudio@icmc.usp.br

Sumário

- Ponteiros
- Ponteiros e Vetores
- Aritmética de Ponteiros

Ponteiros

- Uma variável é armazenada em um certo número de bytes em uma determinada posição de memória.

- Um ponteiro é uma variável que contém o endereço de outra variável.

- Ponteiros são usados para acessar e manipular conteúdos em determinado endereço de memória.

Ponteiros

- Acesso ao endereço de memória da variável:

  &<nome_var>

- Declaração de um ponteiro:

  <tipo> *<nome_var_ponteiro>

- Exemplos de declaração de ponteiros:

```
int *p;
char *q;
float *r,*s;
```

Ponteiros

- Exemplo de atribuição de variáveis e ponteiros:

  p = 0; ⇔ p = NULL;

  p = &i;

  p = (int *) 1776;

- O operador unário & fornece o endereço da variável.

- Logo p = & i atribui o endereço de i à variável p

  p “aponta para” c.

Ponteiros

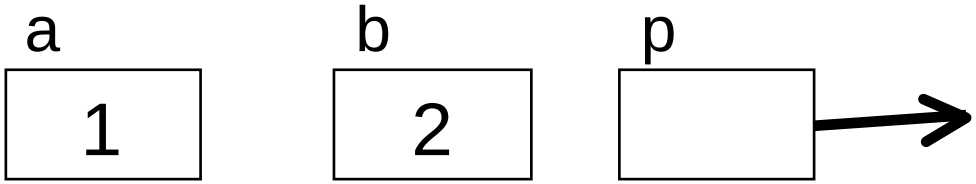
- O operador & só se aplica a objetos na memória: variáveis e elementos de vetor
- Ele não pode ser aplicado a expressões constantes ou variáveis da classe registrador
- São expressões inválidas
  - &3
  - &(k+99)
  - Register v; &v

Ponteiros

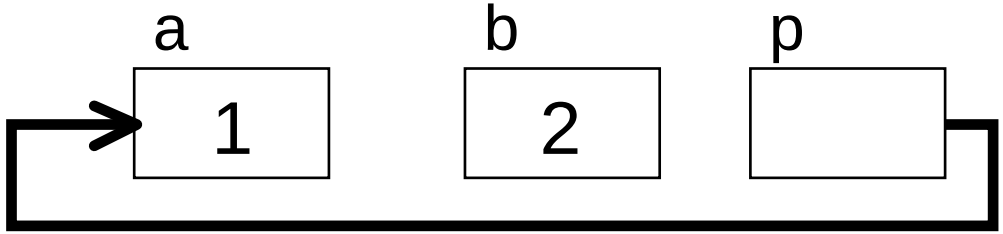
- O operador unário * representa indireção ou deferenciação .
- Se p é um ponteiro, então *p é o valor da variável da qual p é o endereço.
- O valor direto de p é o endereço de memória.
- *p é o valor indireto de p, pois representa o valor armazenado no endereço de memória.

Ponteiros

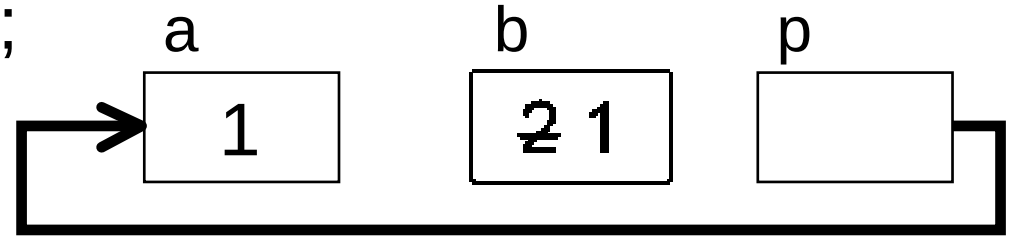
int a=1, b=2, *p;



p=&a



b = *p; ⇔ b=a;



Ponteiros

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i=7, *p = &i;
    printf("%s%d\n%s%p\n", " Valor de i: ", *p,
          "Endereco de i:", p);
    return 0;
}
```

```
Valor de i: 7
Endereco de i: 0028FF18
```

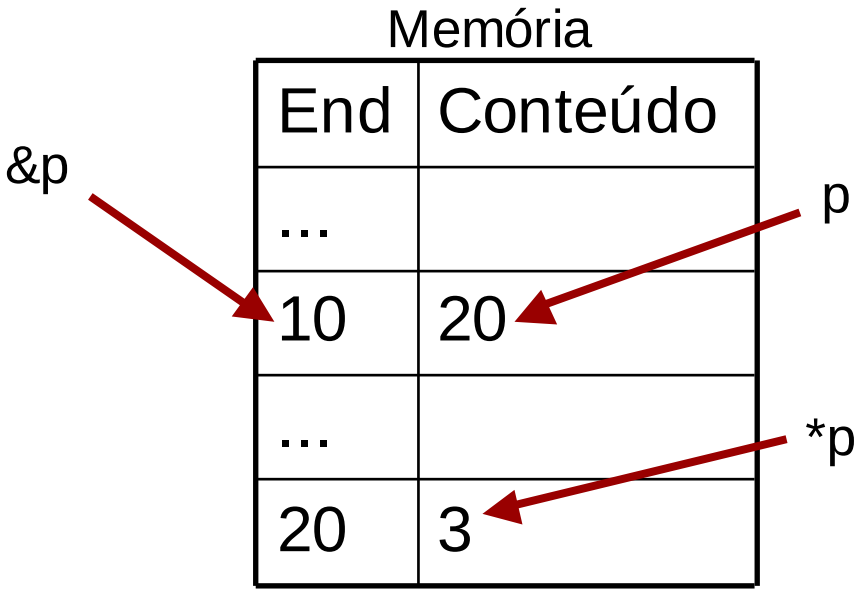
Ponteiros

Declaração e inicialização		
int i=3, j = 5, *p = &i, *q=&j, *r;		
double x;		
Expressão	**Expressão Equivalente**	**Valor**
p == & i	p == (&i)	1
* * & p	* (* (& p))	3
r = & x	r = (& x)	/* ilegal*/
7 * * p / * q + 7	(((7 * (* p))) / (* q)) +7	11
* (r = & j) *= * p	(* (r = (& j))) *= (* p)	15

Cuidado: (7 * * p / * q + 7) ≠ (7 * * p / * q + 7)

Ponteiros

- Considere para *p:



- * * & p
  - ➢&p -> 10
  - ➢*&p -> 20
  - ➢**&p -> 3
- *&p -> p

Ponteiros

- Uma declaração void * <nome_ptr> cria um ponteiro do tipo genérico.
- Conversões durante atribuição entre ponteiros diferentes normalmente eram permitidas no C tradicional.
- Porém, conversões de tipo envolvendo ponteiro não são permitidas no padrão ANSI C.
- No ANSIC, a conversão ocorre apenas se um dos tipos é um ponteiro void ou o lado direito é uma constante 0.

Ponteiros

Declaração	
int *p; float *q; void *v;	
Permitido	**Não permitido**
p=0;	p = 1;
p = (int *) 1;	v = 1;
v = q; v = p;	p = q;
p = (int *) q;	

Ponteiros e Vetores

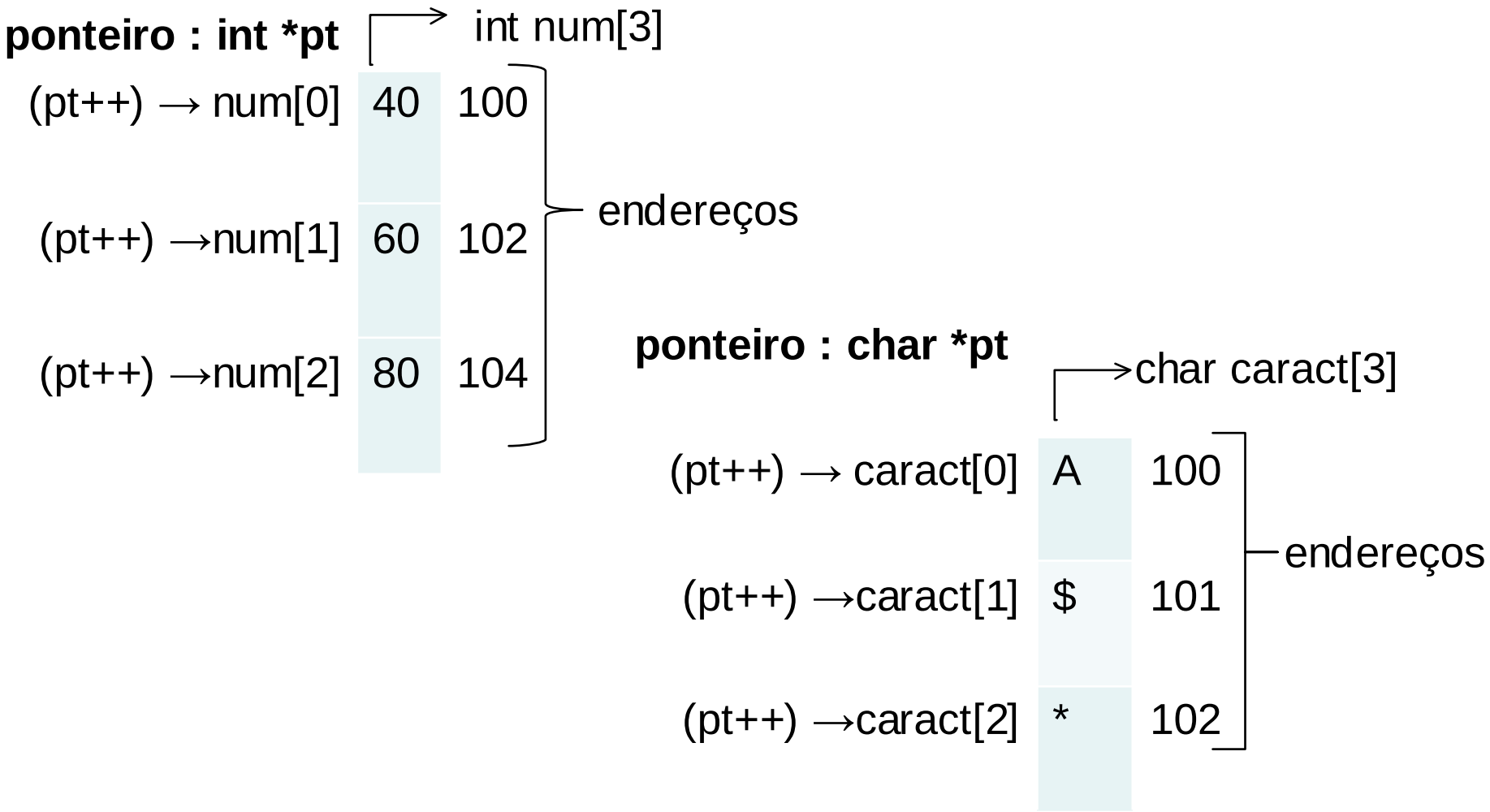
- Um ponteiro pode acessar diferentes endereços.
- Uma determinada posição em um vetor é um endereço ou um ponteiro que está fixo.
- Considere o vetor **a[ ],** onde **a** aponta para a posição i=0. Temos que:

$$\mathbf{a[\ i\ ]} \Leftrightarrow \mathbf{*(a + i)}$$

- Considere o ponteiro **p**, também temos que:

$$\mathbf{p[\ i\ ]} \Leftrightarrow \mathbf{*(p + i)}$$

Ponteiros


ponteiro : int *pt
int num[3]
(pt++) → num[0]
40
100
(pt++) →num[1]
60
102
(pt++) →num[2]
80
104
endereços
ponteiro : char *pt
char caract[3]
(pt++) → caract[0]
A
100
(pt++) →caract[1]
$
101
(pt++) →caract[2]
*
102
endereços

Ponteiros e Vetores

- Incremento/decremento de ponteiros

  int vet[100], *p_vet;

  p_vet = vet;

  p_vet++; *// aponta para o próximo elemento do vetor*

  p_vet--; *// aponta para o elemento anterior do vetor*

  p_vet += 4; *//aponta para a posição atual do vetor + 4;*

OBS: incr. ou decr. qndo vier <u>após</u> o operando é realizado somente após a execução da instrução ter sido completada (o novo valor está disponível somente na próxima instrução), não interessando se há ou não parênteses!

Ponteiros e Vetores

```
int vet[] = {10,11,12,13}, *p_vet, cnt;
p_vet =  vet;
for(cnt=0; cnt<3; cnt++){
      printf("\n %d", *p_vet++);
}

p_vet =  vet;
for(cnt=0; cnt<3; cnt++){
      printf("\n %d", *++p_vet);
}
```

Qual a diferença do resultado impresso pelos dois “printf”?

Ponteiros e Vetores

- Considere as declarações abaixo:

```
# define    N     100
int  a[N], i, *p, sum=0;
```

- Temos que:

p = a ⇔ p=&a[0];

p = a+1 ⇔ p=&a[1];

- Suponha que o vetor a[N] tenha sido inicializado. As rotinas abaixo são equivalentes

```
for (p=a; p < &a[N]; ++p)
   sum += *p;
```

```
for (i=0; i < N; ++i)
    sum += *(a+i);
```

```
p=a;
for (i=0; i < N; ++i)
    sum += p[ i ];
```

Ponteiros e Vetores

- No exemplo anterior, o vetor **a[N]** tem o identificador **a** como um ponteiro constante.

- Logo, as expressões abaixo são ilegais:

  a = p **++a** **a+=2** **&a**

- Não é possível mudar o valor de **a**.

Ponteiros e Vetores

- Podemos passar parte de um vetor para uma função, passando um apontador para o início do subvetor.

  func(&a[4]) $\Leftrightarrow$ func(a+4)

- A declaração dentro da função func() pode ser:

  func(int vet[ ]){...} ou func(int *vet) {....}

- Desde que os limites do vetor sejam obedecidos, pode-se utilizar indexação invertida.

  vet[-1], vet[-2], vet[-3],...

Aritmética de Ponteiros

- pt ± a : pt referencia posições de memória depois (+a) ou antes (-a) da posição atual. O valor **a** é inteiro.
  - Ex: pt+10 ou pt-10

- NÃO é permitido operar dessa forma com **a** sendo um float ou double.

- Outras operações aritméticas NÃO permitidas:
  - pt1+pt2, pt1*pt2, pt1/pt2

Aritmética de Ponteiros

- Operações aritméticas permitidas:
  - pt++, pt--; pt1-pt2 (produz um inteiro).

- Também é possível fazer comparações :
  - pt1>pt2, pt1>=pt2, pt1<pt2, pt1<=pt2, pt1==pt2
  - Essas operações comparam pela posição de memória.
  - Assim pt1 tem posição “mais alta” de memória que pt2, se pt1>pt2.

Aritmética de Ponteiros

- Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   double a[2], *p, *q;
   p=a;                                  /* aponta para a[0]*/
   q=p+1;                               /* aponta para a[1], ou seja, q=&a[1]*/
   printf("%d\n", q-p);                          /* exibe valor 1 */
   printf("%d\n", (int) q - (int) p);            /* exibe valor 8 */
   return 0;
}
```

Exercício I

- Escreva um programa que:
  - Inicie uma matriz 20x20 com valores aleatórios no intervalo [10,-10].
  - Determine o maior e o menor valor dessa matriz.
  - Determine a quantidade de valores negativos dessa matriz.
- Funções devem ser utilizada para cada tarefa.
- Ponteiros devem ser utilizados sempre que possível.

Exercício II

- Faça um programa que simule uma pilha. O tamanho máximo da pilha é 100 e apenas valores inteiros podem ser armazenados. As seguintes operações devem estar disponíveis:
    - Imprimir
    - Inserir
    - Retirar

Referências

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

Kelley, A.; Pohl, I., *A Book on C*: programming in C. 4ª Edição. Massachusetts: Pearson, 2010, 726p.

Kernighan, B.W.; Ritchie, D.M. C, *A Linguagem de Programação*: padrão ANSI. 2ª Edição. Rio de Janeiro: Campus, 1989, 290p.

FIM Aula 14