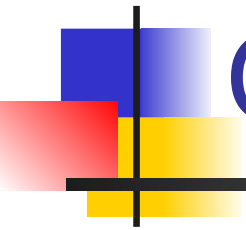


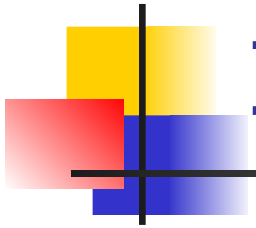
SCC0221 - Introdução à Ciência de Computação I

Prof.: Dr. Rudinei Goularte

(rudinei@icmc.usp.br)

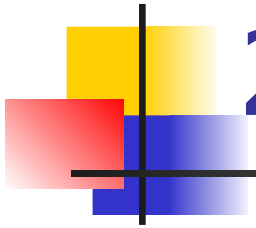
Ponteiros em C

Instituto de Ciências Matemáticas e de Computação - ICMC
Sala 4-229



1. Introdução

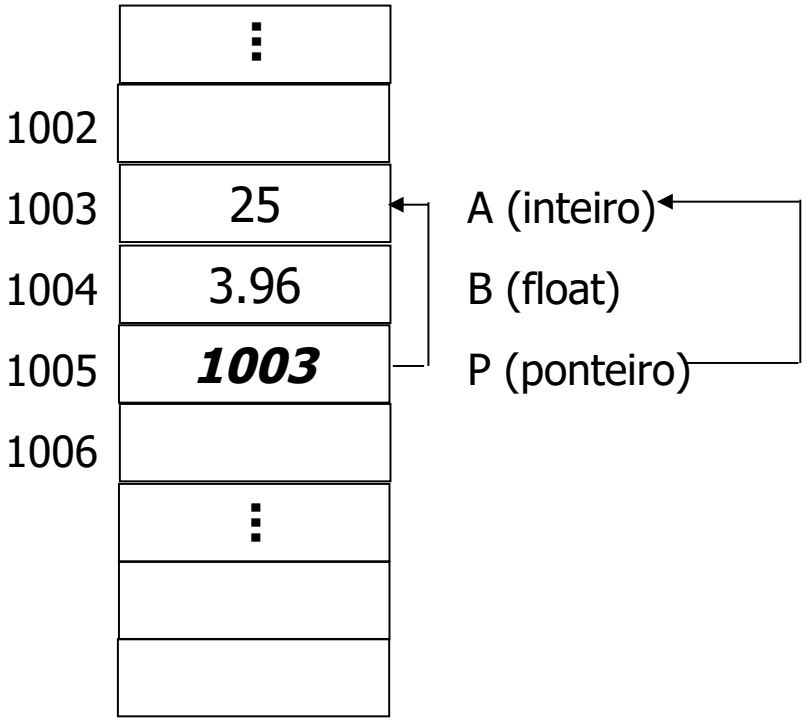
- Por quê ponteiros são importantes?
  - Ponteiros fornecem meios para funções modificarem seus argumentos.
  - São usados para alocar memória dinamicamente.
  - Pode aumentar a eficiência de certas rotinas.
- **O Entendimento e uso correto de ponteiros em C é crítico!**



2. O que é um ponteiro?

- **É uma variável que contém um endereço de memória.**
  - Normalmente, esse endereço é a posição de outra variável na memória.
  - Dizemos que um ponteiro “aponta” para uma variável.

2. O que é um ponteiro?



3. Declaração de ponteiros em C

- Forma geral: **tipo *identificador;**

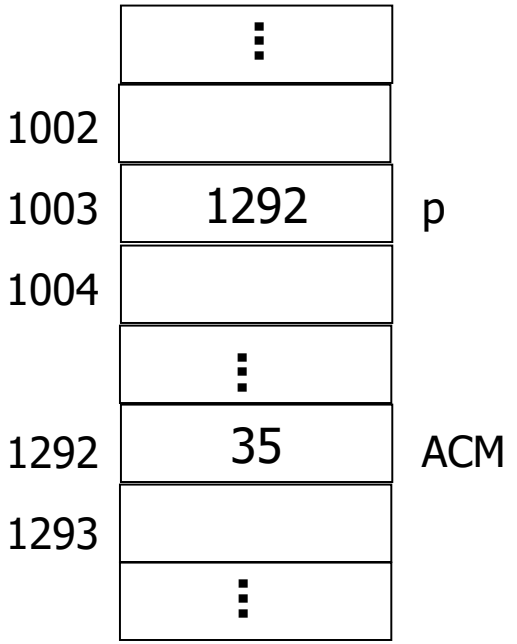
  - **tipo**: qualquer tipo válido em C.
  - identificador: qualquer identificador válido em C.
  - *: símbolo para declaração de ponteiro. Indica que o **identificador** aponta para uma variável do tipo **tipo**.

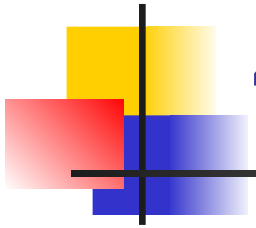
4. Operadores de ponteiros

- &: Endereço.
- Operador unário. Devolve o endereço de memória de seu operando.
  - Ex.:

```
int *p, acm = 35;
p = &acm; /*p recebe "o endereço de"
                    acm*/
```

  - O valor de p é 1292



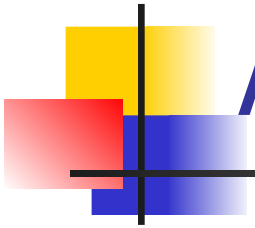


4. Operadores de ponteiros

- *: Dereferência.
- Operador unário. Devolve o valor da variável apontada.
  - Ex.:

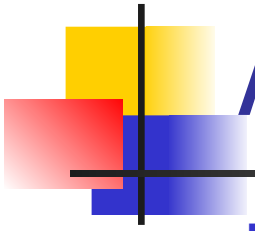
```
int *p, q, acm = 35;
p = &acm;
q = *p;     /*   q recebe o conteúdo da variável
                 "no endereço" p   */
```

  - O valor de q é 35



Atenção!!!!

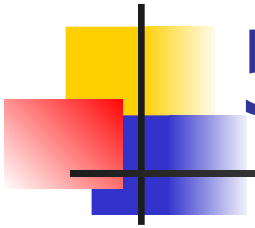
- Não confundir os operadores de ponteiros & (“o endereço de”) e * (“dereferência”) com o operador AND bit a bit e com o sinal de multiplicação.
  - Os operadores de ponteiros têm precedência maior que todos os operadores aritméticos, exceto o menos unário.



Atenção 2!!!!

- Tecnicamente, qualquer ponteiro pode apontar para qualquer posição de memória.
  - O tipo do ponteiro pode ser importante. Por exemplo em aritmética de ponteiros.

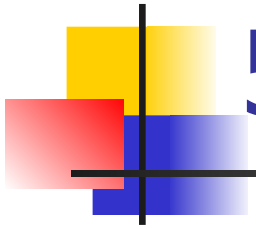
```
float x = 3.5, y = 0.96;
int *p;
p = &x;   /* p (int *) aponta p/ um float!!!*/
y = *p;   /*erro*/
```



5. Expressões com ponteiros

- Atribuição: igual a qualquer variável.

```
int x=8, *p1, *p2;
p1 = &x;
p2 = p1;
printf (“%d %d”, p1, *p1);
printf (“%d %d”, p2, *p2);
```

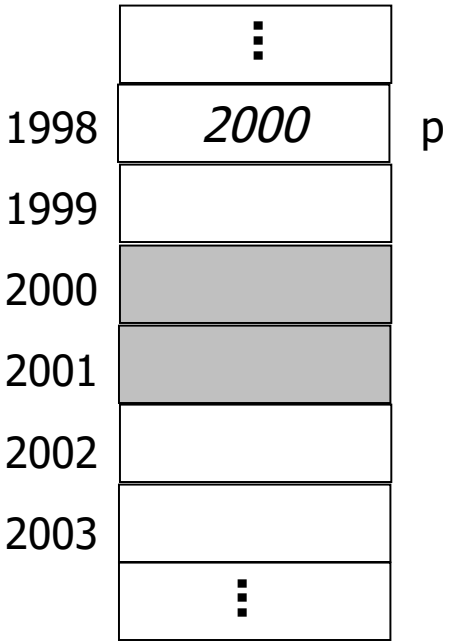


5. Expressões com ponteiros

- Aritmética de Ponteiros
  - 5 operações apenas:
    - Incremento, decremento, comparação, adição e subtração.
  - Cada incremento (ou decremento) aumenta (ou diminui) o valor do ponteiro em quantidade de bytes correspondente ao tipo base.

5. Expressões com ponteiros

- Aritmética de Ponteiros
- Ex. incremento/decremento:



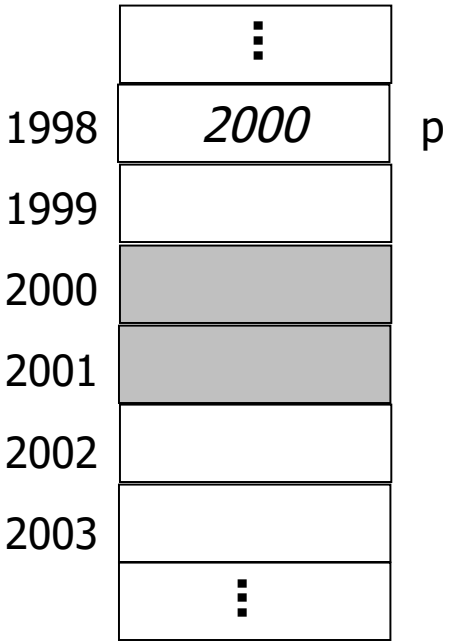
inteiro de 2 bytes:

```
int *p;
...
p++;
/*p--;*/
```

o valor da variável p agora é 2002

5. Expressões com ponteiros

- Aritmética de Ponteiros
- Ex. adição:



inteiro de 2 bytes:

```
int *p, *q;
...
p = p + 2;
```

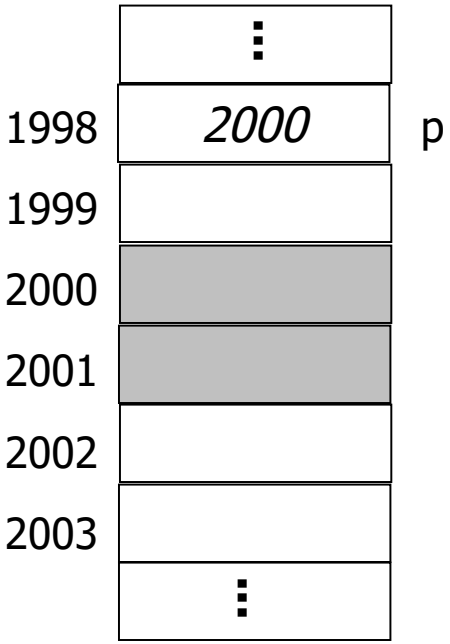
- O valor da variável p agora é 2004.

- Na Adição, o outro operando além do ponteiro deve ser uma **expressão** inteira, mas não outro ponteiro!!

```
p = p + q; /*Erro*/
```

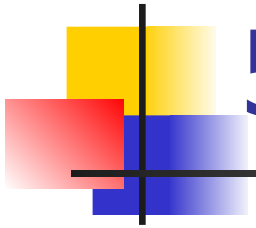
5. Expressões com ponteiros

- Aritmética de Ponteiros
- Ex. subtração:



inteiro de 2 bytes:
int *p, *q, x;
...
p = p - 2;
- O valor da variável p agora é 1996.
...
x = p – q;

- A Subtração entre 2 ponteiros retorna a distância em bytes ente eles.



5. Expressões com ponteiros

- Comparação entre ponteiros é possível.
  - Está-se comparando posições de memória.
  - Aplicação em vetores, matrizes, EDs como pilhas, etc.

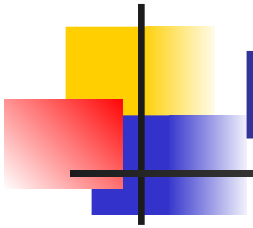
6. Indireção múltipla

- Também conhecida como ponteiros para ponteiros.
- float **q;
  - q é um ponteiro para um ponteiro float.

```
float x = 5.7, *p, **q;

p = &x;
q = p; /*erro*/
q = &p;
printf ("%d",**q);
```

- Indireção pode ser levada a qualquer dimensão
  - int *********ptr;

7. Erros comuns com ponteiros



- “Ponteiros são uma benção que pode trazer problemas” (Herbert Schildt).
  - Operações com ponteiros irregulares podem levar o programador a procurar o erro no lugar errado.
    - Erro pode aparecer mais adiante na execução.
      - Evidências do erro.
      - Horas perdidas em depuração.
  - Invasão de memória.

7. Erros comuns com ponteiros

- Ponteiro não inicializado.
  - Erro mais comum.

```
int x, *p;
x = 10;
*p = 25; /*antes, deveria haver: p = &x; */
```

  - p não aponta p/ um endereço válido.
  - Programas pequenos podem mascarar esse erro.
  - Esse tipo de erro pode "derrubar" o programa ou até o Sistema Operacional.
  - **Boa prática de progr.: inicializar todo ponteiro!**

7. Erros comuns com ponteiros

- Ponteiro inicializado de modo errado.

```
int x, *p;
x = 10;
p = x;  /*errado*/
printf("%d",*p);
```

  - Não irá imprimir o valor de x.
  - p = x; está errado. p deve conter um endereço e não um valor.
  - O correto é p = &x;

7. Erros comuns com ponteiros

- Comparação entre ponteiros que não apontam para a mesma variável.

```
char s[80], y[80];
char *p1, *p2;

gets(s); gets(y);
p1 = s; p2 = y;
if (p1 > p2)  ...
```

  - Comparações de ponteiros comparam posições de memória.
  - Não sabemos onde p1 e p2 estão alocados.

7. Erros comuns com ponteiros

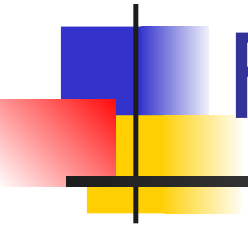
- Reinicialização em iterações.

```
char *p, s[80];

p1 = s; /*erro*/
do{
   gets(s);
   while (*p1) printf("%c", *p1++);
}while(strcmp(s,"fim");
```

7. Erros comuns com ponteiros

- Reinicialização em iterações.

```
char *p, s[80];

do{
   p1 = s;
   gets(s);
   while (*p1) printf("%c", *p1++);
}while(strcmp(s,"fim");
```



Fim.