ponteiros

INF1005 -- Programação I -- 2016.1
Prof. Roberto Azevedo
razevedo@inf.puc-rio.br

ponteiros

tópicos

- o que são ponteiros
- operadores usados com ponteiros
- passagem de ponteiros para funções

referência

- Capítulo 4 do livro pp.45-51

recapitulando...

variáveis locais

- existem apenas no bloco onde foram declaradas
- exemplo: variável raio existe apenas dentro da função area.

```
double area ()
{
   double raio = 1.0;
   return 2 * 3.14 * raio;
}

int main(void)
{
  /* variável raio não pode ser acessada aqui*/
}
```

recapitulando...

parâmetros

- podem ser interpretados como variáveis locais criadas no início da função e inicializadas com os valores recebidos na chamada da função

```
double area (double raio)
{
   /* é similar a "double raio = 1.0;" */
   return 2 * 3.14 * raio;
}

int main(void)
{
  printf("A area vale %f\n", area(1.0));
}
```

algumas perguntas...

Uma função em C pode retornar dois ou mais valores?

R. NÃO

Uma função em C pode acessar (ler ou alterar) variáveis locais de outras funções?

R1. Não diretamente...

R2. Somente se a função tiver os **endereços** dos valores a serem alterados.

Lembrando: **scanf("%d", &num)**

Por que usar ponteiros?

Em C, funções recebem parâmetros **por valor** (cópia)

se um valor é modificado dentro da função, quando a função termina (desempilha), os valores originais se mantêm inalterados

```
void somaprod(int a, int b, int soma, int prod)
{
    soma = a+b; prod = a*b;
}
int main(void) {
    int s, p;
    somaprod (3, 5, s, p);
    printf("soma: %d \nproduto: %d\n", s, p);
    return 0;
}
```

2
3
1
4

04_17_func_par_valor.c

1. após decl. *main*

main	**p**	**?**
	s	**?**

2. chamada de *somaprod*

somaprod	**prod**	**?**
	soma	**?**
	b	**5**
	a	**3**
main	p	?
	s	?

3. após os cálculos

somaprod	**prod**	**15**
	soma	**8**
	b	5
	a	3
main	p	?
	s	?

4. na volta para a *main*

main	p	?
	s	?

```
soma: 751351
produto: -267762
(lixo)
```

Por que usar ponteiros?

funções podem receber **endereços** como parâmetro

se uma função conhece o endereço de uma posição de memória, ela pode modificar o conteúdo dessa posição

```
void somaprod(int a, int b, int* soma, int* prod)
{  /* muda o valor do que está no endereço respectivo */
   *soma = a+b; *prod = a*b;
}
int main(void) {
   int s, p;
   somaprod (3, 5, &s, &p); /* passa valores 3 e 5 e endereços de s e p */
   printf("soma: %d \nproduto: %d\n", s, p);
   return 0;
}
```

2 3 1 4

04_19_func_par_ref.c

1. após decl. *main*

main	p	?	108
	s	?	104

2. chamada de *somaprod*

somaprod	prod	108	124
	soma	104	120
	b	5	116
	a	3	112
main	p	?	108
	s	?	104

3. atrib. após cálculos

somaprod	prod	108	124
	soma	104	120
	b	5	116
	a	3	112
main	p	15	108
	s	8	104

4. na volta para a *main*

main	p	15	108
	s	8	104

5. após *printf*

```
soma: 8
produto: 15
```

ponteiro

variável que contém o endereço de uma variável

- às vezes são a única forma de expressar uma computação
- costumam levar a códigos mais compactos (às vezes mais obscuros)
- é fácil criar ponteiros que apontam para um lugar inesperado
  - Fonte de bugs: corrupção de memória, falha de segmentação etc.

operadores unários

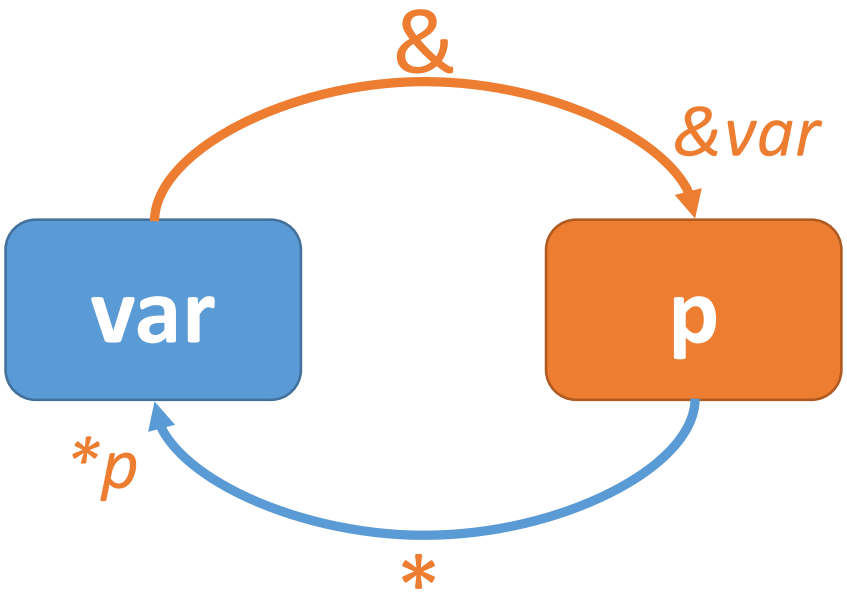
endereço-de (&)

- avalia para o endereço de um objeto de memória (variável ou elemento de vetor)
- não pode ser aplicado a constantes ou expressões
- exemplos: &x, &v[3]

conteúdo-de (*)

- avalia para o objeto de memória (variável ou elemento de vetor) para o qual o ponteiro aponta
- exemplo: *p, *(v+3)

operadores unários



declaração de ponteiros

variável

int *ip;

- Declara uma variável **ip** do tipo "**ponteiro para int**", i.e., cujo conteúdo é o endereço de um objeto de memória do tipo int
- Ou seja, ***ip** avalia para **int**

parâmetro ou retorno de função

char * func (double *);

- Declara uma função que recebe um ponteiro para double e retorna um ponteiro para char
- Ou seja, *(func (pd)) avalia para char e *pd avalia para double

exemplos

```
int x = 1, y = 2;
int *ip; /* declara ponteiro para int */
ip = &x; /* ip aponta para x */
y = *ip; /* y <- x */
*ip = 0; /* x <- 0 */
```

exemplo

EX. 01

```
int main ( void )
{
    int a=5;
    int *p;
    p = &a;
    *p = 2;
    (*p)++;
    printf("%d ", a);
    return 0;
}
```

Qual é a saída do programa?

importante

- Se **ip** aponta para a variável **x**, então ***ip** é equivalente a **x**
- Ou seja, ambos avaliam sempre para a mesma coisa
- Ou seja, podemos substituir um pelo outro em qualquer contexto

exemplo

```
int x = 10;
int *ip = &x;
y = *ip + x;   /* qual o valor de y? */
```

precedência

- em geral, & e * tem precedência sobre operadores aritméticos
- na dúvida, **use parênteses**

restrição

- Um ponteiro só pode apontar para variáveis ou elementos de vetores do seu tipo (exceção: void *)
- Ou seja, um ponteiro para int só pode apontar para objetos de memória do tipo int (só podemos atribuir a ele endereços de int)
- Exemplo:

```
int *p = NULL;
int *q = &p;
printf (“%p”, *q);
```

exemplo 1

- faça uma função para trocar o valor de duas variáveis

exemplo 1

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

int main (void)
{
        int m, n;
        m = 10;
        n = 20;
        printf ("m = %d, n = %d\n", m, n);
        troca(&m, &n);
        printf ("m = %d, n = %d\n", m, n);

        return 0;
}
```

exemplo 2

- Faça uma função para calcular as raízes de uma equação do segundo grau. A função deve ter o seguinte protótipo:

```
int equacao_quadratica (double a, double b, double c,  double *raiz1, double *raiz2);
```

exemplo 2

```
int equacao_quadratica (double a, double b, double c,
                        double *raiz1, double *raiz2)
{
        double delta;
        delta = b*b - 4*a*c;

        if (delta < 0) {
                return 0;
        }
        else if (delta == 0.0) {
                *raiz = -b/(2*a);
                return 1;
        }
        else {
                *raiz1 = (-b - sqrt(delta))/ (2*a);
                *raiz2 = (-b + sqrt(delta))/ (2*a);
                return 2;
        }
}
```

exercício 1

escreva uma função que receba um número inteiro *n* e dois endereços para números reais, leia *n* números reais, atribua ao conteúdo dos ponteiros os valores mínimo e máximo dos *n* números e retorne sua média aritmética. Escreva também um programa para testar sua função.

exercício 1

```
#include <stdio.h>
float media(int n, float* min, float* max) /* aqui, min e max são ponteiros para inteiros */
{
   _____________________________________________
   _____________________________________________
   _____________________________________________
   _____________________________________________
   _____________________________________________
   _____________________________________________
   _____________________________________________
}

int main(void)
{
  float min, max, med = media(5, &min, &max); /* aqui, min e max são inteiros */
  printf("min: %f, max: %f, media: %f\n", min, max, med);
  return 0;
}
```

```c
#include <stdio.h>
float media(int n, float* min, float* max) /* aqui, min e max são ponteiros para inteiros */
{
   int i;
   float valor, soma;
   if (n <= 0) /* se n não for positivo, sai retornando zero */
   {
     *min = 0; *max = 0;
     return 0.0F;
   }
   scanf("%f", &valor); /* lê o primeiro valor */
   *min = *max = soma = valor; /* inicializa as variáveis com o primeiro valor lido */
   for (i=2; i <= n; i++) /* lê os próximos (n-1) valores */
                {
      scanf("%f", &valor);
      if (valor < *min) *min = valor; /* verifica se valor lido é o novo mínimo */
      if (valor > *max) *max = valor; /* verifica se valor lido é o novo máximo */
         soma += valor; /* soma o valor lido para o cálculo da média */
   }
   return soma/n;
}
int main(void)
{
   float min, max, med = media(5, &min, &max); /* aqui, min e max são inteiros */
   printf("min: %f, max: %f, media: %f\n", min, max, med);
   return 0;
}
```

dúvidas?