estudar.com.vc

Mac em C

Resumo e Exercícios P2

Resumo Teórico

Vetores

Declaramos vetores da seguinte forma:

```
int vetor[50];
```

onde 50 é exemplo de um número inteiro e constante.

Podemos também fazer um #define no início do programa

```
#define TAMANHO 45
int main()
{
   int vetor[50];
   int vetor2[TAMANHO];
}
```

Para inicializar vetores pequenos podemos digitar seu conteúdo:

```
int vetor3[4] = {1, 2, 3, 4};
```

Funções

Funções são blocos de código que a partir de um parâmetro, produzem um resultado, análogas às funções matemáticas. O tipo do resultado da função caracteriza o tipo da função: se temos um int como resposta, declaramos int função(). Toda função não-void tem um return, comando que retorna o resultado do que foi feito dentro da função.

```
int funcao (int a, int b)
{
        <código>

}
```

Normalmente fazemos funções quando um bloco específico de código desempenha uma função específica que pode ser utilizada mais vezes, ou quando temos muito código e o separamos em funções para melhor entendimento.

Para chamar uma função, é igual ao que fazemos para printf e scanf, afinal estas também são funções, porém até agora não utilizamos os resultados dessas funções (sim, essas funções têm return!). Para armazenar o resultado, basta atribuir a uma variável uma chamada de função:

```
int a;
int b, c;
a = função(b, c);
```

Ponteiros e funções void

Ponteiros são variáveis que guardam endereços da memória. Gosto de comparar ponteiros a envelopes de cartas. Os ponteiros são os envelopes, e as cartas são os conteúdos para os quais os ponteiros apontam.

&p -> envelope

p-> carta

O operador & significa “o endereço de”. Todas as variáveis têm um endereço, e utilizamos ponteiros para guardá-los.

```
int a;
int* p = &a;
```

Quando vamos enviar uma carta, compramos o envelope e compramos o papel. Não tem como escrevermos uma carta num envelope, precisamos do papel. Eventualmente, podemos escrever uma carta em casa e depois comprar o envelope para colocar a carta dentro. Nesse momento, é isso que aprenderemos

a fazer. Para a P3, teremos o envelope, e aí depois escreveremos as cartas. O legal do envelope é que podemos ter vários papéis dentro, quando a carta é muito comprida. Mas o envelope continuará sendo um envelope.

Saindo um pouco da analogia, aprenderemos nesse momento a igualar o ponteiro a um endereço de uma variável já existente, mas no futuro aprenderemos a criar o ponteiro e em seguida alocar um espaço na memória para o qual esse ponteiro aponta. O ponteiro pode economizar muito espaço quando falamos de vetores e matrizes. Toda vez que passamos uma variável como parâmetro para uma função, o que acontece internamente é que uma cópia da variável é passada. Ou seja, ao passar um vetor como parâmetro, todos os valores armazenados são passados como uma cópia, ao passo que se usarmos um ponteiro, somente um endereço é passado, e assim temos mais dinamicidade. Um último diferencial do ponteiro é que a variável para qual ele aponta é alterada independentemente do contexto, muitas vezes dispensando o return das funções, deixando espaço para que a função mexa em várias variáveis e retorne códigos de status ou erros ao invés de valores propriamente ditos.

Nesse contexto é que usamos funções do tipo void, em que as variáveis parâmetro são ponteiros.

Strings e char

Char é o tipo de variável que utilizamos para tratar letras do alfabeto. Porém, em C, char nada mais é que um int de tamanho reduzido (aliás o menor tipo de variável possível em C), indo de -127 a 127. Podemos imprimir um caractere através do modificador %c no printf. Além disso, se estiver no intervalo mencionado acima, podemos também imprimir a partir de um valor de uma variável de tipo int.

A leitura de um conjunto de caracteres, uma string, é feita com o modificador %s no scanf, e o segundo parâmetro tem que ser um vetor grande o suficiente para conter a string. Além disso, é bom também saber que esse vetor de caracteres sempre terminará com o símbolo \0, para indicar o término da string.

A biblioteca string.h é muito útil para trabalharmos com strings, contendo vários métodos que poupam bastante trabalho:

strlen(char *str) -> retorna o tamanho da string

strcpy(char *dest, const char *src) -> copia a string src para a string dest

strcat(char *dest, const char *src) -> coloca a string src no final da dest.

Outras funções úteis podem ser vistas pesquisando o nome da biblioteca na internet.

Switch case

Uma forma mais organizada de fazer uma seleção condicional, o switch é como uma lista de possibilidades de valor de uma variável e a ação resultante de cada uma delas. No final temos o default, que é o caso em que nenhuma das outras possibilidades se aplicam. Recomendado quando há muitos else if encadeados.

```
switch(selecao)
  {
    case 1:
      printf("caso 1");
      break;
    case 2:
      printf("caso 2");
      break;
    default:
      printf("nenhum caso");
```

```
        break;
    }
```

É importante frisar que não é permitido comparações lógicas no switch. Apenas testamos valores absolutos. Portanto, isso significa que podemos ter ali no case caracteres, valores reais (pseudo-reais) e strings.

Dicas

vetor[i++] = j; -> atribui j pra posição e soma 1 no i

vetor[++i] = j; ->Soma 1 no i e atribui j pra posição resultante

Escrever (int) na frente do nome de uma variável faz com que o compilador leia a variável com int, mesmo que tenha sido declarada como double por exemplo. Nesse caso em específico, (int) a para um a = 4.36 vai resultar em 4. Ou seja, equivale a truncar o número. Essa prática é chamada de casting.

Exercícios

1. Exercício de Prova

P2 2016, questão 1

Faça um programa em C com uma função com assinatura *int pega(int N, int i)* que devolve o dígito de potência *i* de *N*. O programa principal deve solicitar os valores naturais para *N* e *i* e, usando o valor devolvido pela função *pega*, imprimir o dígito i. *Exemplos:* Para N = 1023 e i = 2, deve-se imprimir 0. Para N = 1023 e i = 0, deve-se imprimir 3.

2. Exercício de Prova

P2 2016, questão 2

Faça um programa em C com uma função com assinatura *float aproxima(float x, float err)* que devolve o valor da expressão $\sum_{k=0}^{n} termo_k$, sendo n o primeiro valor tal que o termo $|termo_k| = |(-1)^k x^{2k+1} / (3 \times 2k!)| < err$ (com $err > 0$). No programa principal deve-se solicitar que o usuário digite os valores para *x* e *err* e, usando o valor devolvido por *aproxima*, imprimir o valor do somatório.

3. Exercício de Prova

P2 2016, questão 3

Faça um programa em C que, dado um natural N, imprima seu dígito de controle definido pelo seguinte processo: R: compute a soma s dos dígitos de N se s < 10, imprima s, senão repita o passo R usando N com o valor de s.
Exemplos: Para N = 6409, computa 6 + 4 + 0 + 9 = 19, computa 1 + 9 = 10, computa 1 + 0 = 1, então imprime 1.

4. Exercício de Prova

P2 2016, questão 4

Faça uma função com assinatura *void simplifica(int num, int den, int *rn, int *rd)* que simplifica o racional *num/den* (ambos naturais, *den* > 0). *Exemplo: num = 3, den = 21 => devolve rn = 1 e rd = 7*

5. Exercício de Prova

P2 2016, questão 5

Faça um programa em C que: leia um natural *n* > 0 e *n* pares de *num* e *den* (numeradores e denominadores) em um único vetor, sendo *num* em posições pares do vetor e *den* em ímpares (sequenciais). Usando a função *void simplifica(int num, int den, int *rn, int *rd)* anterior (mesmo não tendo feito pode supor feita),

computar a soma dos racionais e imprimir sua soma também na forma racional reduzida.

Exemplo 1: Para *n* = 3 e os racionais 3/4, 2/5 e 1/7, o vetor de dados terá

3	4	2	5	1	7

E a resposta será 181/140 (pois 3/4 + 2/5 + 1/7 + 181/140).

Exemplo 2: Para *n* = 4 os racionais 1/4, 1/4, 1/4, 1/4, o vetor de dados será

1	4	1	4	1	4	1	4

E a resposta será 1/1.

6. Exercício de Prova

P2 2015, questão 1

Escreva um programa na linguagem C que lê como entrada um numero real *épsilon* > 0 e exibe como saída um número real que é uma aproximação de π calculada por meio da seguinte série infinita:

$$\frac{\pi}{6} = \frac{1}{2} + \left(\frac{1}{2}\right)\frac{1}{3}\left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\frac{3}{4}\right)\frac{1}{5}\left(\frac{1}{2}\right)^5 + \left(\frac{1}{2}\frac{3}{4}\frac{5}{6}\right)\frac{1}{7}\left(\frac{1}{2}\right)^7 + \left(\frac{1}{2}\frac{3}{4}\frac{5}{6}\frac{7}{8}\right)\frac{1}{9}\left(\frac{1}{2}\right)^9 + \cdots$$

Inclua na aproximação todos os termos da série até o primeiro termo de valor menor do que *epsilon*. Inclua também na soma esse último termo calculado.

Descubra o que muda de um termo para outro na soma e use essa informação para escrever sua função (ou seja, para calcular um termo da soma, você deve reaproveitar valores envolvidos no cálculo do termo anterior).

7. Exercícios de prova

P1 2002, questão 1

Para um número real $x \geq 0$, denotamos por $\lfloor x \rfloor$ seu chão, que é o maior inteiro menor ou igual a x. Por exemplo, temos $\lfloor 3 \rfloor = 3$ e $\lfloor 4.73 \rfloor = 4$.
Dado um número real $a_0 \geq 1$, podemos considerar a sequência de números a_0, a_1, a_2, . . . tal que

$$a_{k+1} = \begin{cases} \dfrac{a_k}{2} & se\ \lfloor a_k \rfloor \text{ é } par, \\ \dfrac{3a_k + 1}{4} & se\ \lfloor 3a_k + 1 \rfloor \text{é múltiplo de } 4, \\ \dfrac{3a_k + 1}{2} & caso\ contrário \end{cases}$$

para $k \geq 0$.
Por exemplo, começando com $a_0 = 13$, obtemos a sequência

13, 10, 5, 4, 2, 1, 1 . . . .

Observe que após a ocorrência do número 1, todos os demais números da sequência são iguais a 1. Começando com $a_0 = 5/2$, obtemos

5/2, 5/4, 19/16, 73/64, 283/256, 1105/1024, 4339/4096, ...

Temos $4339/4096 \approx 1.05932$. Se você continuar a sequência, perceberá que os números vão ficando cada vez mais próximos de 1.
Chamamos a *sequência* $(a_k)_{k\geq 0}$ *de sequência de Collatz real* com *valor inicial* a_0.

a) Escreva uma função na linguagem C de protótipo

int collatz(double x, double eps);

que, dados números reais $x \geq 1$ e $eps > 0$, calcula a sequência de Collatz $(a_k)_{k\geq 0}$ de valor inicial $a_0 = x$ e devolve o MENOR n tal que $|a_n - 1| < eps$.
Por exemplo, se $x = 5/2$ e $eps = 0.06$, sua função deve devolver 6, já que $a_6 = 4339/4096 \approx 1.059$ e os valores anteriores são maiores ou iguais a $1105/1024 \approx 1.079$.

b) Você deve escrever um programa na linguagem C para calcular o maior valor de sua função do item (a) para o conjunto de números $x_r = 1 + r\delta$ com $0 \leq r \leq N$ inteiro, onde $\delta = (u - 1)/N$.
Assim, seu programa deve ler números reais $u \geq 1$ e $eps > 0$ e um número inteiro $N \geq 1$ e imprimir na tela o número

max{ collatz(x_r, *eps*) : 0 ≤ r ≤ N é inteiro }.

Você pode usar a função do item anterior mesmo que não a tenha escrito.

8. Exercícios de prova

P2 2015, questão 3

Nesta questão você vai implementar uma função na linguagem C para a soma de números de ponto flutuante. Neste problema vamos supor que a macro (= constante) MAX contenha um valor tal que uma variável do tipo int consiga comportar números de valor absoluto menor ou igual a $10 * \text{MAX}$. Nosso objetivo é sempre trabalhar com mantissas de valor absoluto menor ou igual a MAX. Um número de ponto flutuante é representado por um par (m, e) de números inteiros, sendo m a mantissa e e o expoente. O par (m, e) representa o número

$$m \times 10e.$$

Um número está normalizado se sua mantissa não possui zeros à direita. Por exemplo, o número (12300, 5) não está normalizado, já o número (123, 7) está normalizado. Você deve escrever uma função de protótipo

int soma(int *res m, int *res e, int x m, int x e, int y m, int y e);

que recebe dois números de ponto flutuante normalizados (x_m, x_e) e (y_m, y_e) com $|x\_m|, |y\_m| \leq$ **MAX** e coloca em $*res\_m$ e $*res\_e$ a mantissa e o expoente do resultado NORMALIZADO da soma dos dois números recebidos. A função deve devolver ZERO se ocorreu perda de precisão durante a soma e um valor DIFERENTE DE ZERO caso contrário. LEMBRE-SE: devemos ter ao final $|*res\_m| \leq$ **MAX**. Para escrever sua função, lembre-se de que:

- Para fazer a soma, precisamos igualar os expoentes dos números (caso eles não sejam iguais), mas sempre com o cuidado de manter a maior quantidade possível de dígitos de precisão no resultado.
- Com os números com expoentes iguais, podemos somar suas mantissas e normalizar o resultado.
- Se ao final o valor absoluto da mantissa for maior que MAX, devemos dividí-la por 10 e aumentar o expoente para obter uma mantissa de valor absoluto menor ou igual a MAX.

Neste exercício, você não precisa se preocupar com a ocorrência de overflow no expoente de um número.
EXEMPLO: Suponha que MAX = 999. Vamos somar a = (99, 5) com b = (587, 3). Primeiro diminuímos o expoente de a (que é o número de maior expoente), obtendo (990, 4); note que não podemos diminuir mais o expoente. Depois, aumentamos o expoente de b até obter expoente 4, obtendo (58, 4) com perda de precisão. Depois, fazemos 990 + 58 = 1048 > 999, logo temos que consertar a mantissa e perder precisão novamente. Assim, obtemos o resultado (104, 5) com perda de precisão.

9. Exercícios de prova

P2 2015, questão 4

Dizemos que o par de números (p, q) é um par de primos gêmeos se q = p + 2. Por exemplo, (101, 103) é um par de primos gêmeos. A função de protótipo *void devolvePrimosGemeos(int i, int *p1, int *p2);* devolve em *p1 e *p2 o i-ésimo par de primos gêmeos, onde (3, 5) é o primeiro par de primos gêmeos, (5, 7) é o segundo, (11, 13) é o terceiro, e assim por diante. A seguir, são apresentadas 4 diferentes implementações da função *devolvePrimosGemeos*. Em cada uma delas, você deve indicar se o código está correto ou incorreto. Ao lado de toda implementação que indicar como incorreta, escreva brevemente sobre o(s) erro(s) encontrado(s). Obs.: (i) O valor da questão será anulado caso todas as respostas forem marcadas como “Correto”. (ii) Você deve supor que a função de protótipo *int ehPrimo(int num);* (que devolve 1 quando num é primo e 0 caso contrário) já existe.

a)
```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
  int k = 0, p;
  for (p = 3; k < i; p++) {
    if (ehPrimo(p) && ehPrimo(p+2))
      k++;
  }
  p1 = p;
```

Correto ( )
Incorreto ( )

```
        p2 = p+2;
    }
```

b)
```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
    int k = 0, p;
    for (p = 2; k < i; p++)
      if (ehPrimo(p) && ehPrimo(p+2))
        k++;
    *p1 = p-1;
    *p2 = p+1;
}
```

Correto ( )
Incorreto ( )

c)
```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
      int k = 1, p = 0;
      while (k <= i) {
        p++;
        if (ehPrimo(p) && ehPrimo(p+2))
          k++;
      }
      *p1 = p;
      *p2 = p+2;
  }
```

Correto ( )
Incorreto ( )

d)
```
void devolvePrimosGemeos(int i, int *p1, int *p2) {
      int k = 0, p = 1;
      while (k <= i) {
        p+=2;
        if (ehPrimo(p) && ehPrimo(p+2))
          k++;
      }
      *p1 = *p;
      *p2 = (*p)+2;

    }
```

Correto ( )
Incorreto ( )

Gabarito

1.

```c
#include <stdio.h>

int pega (int  N, int i)
{
   while (i > 0 && N > 0){
      N /= 10;
      i --;
   }
   if (N > 0)
      return (N%10);
    else
      return -1;
}

int main ()
{
   int N, i;
   printf("Digite N e i: ");
   scanf("%d%d", &N, &i);
   if (pega (N, i) >= 0)
      printf("%d", pega(N, i));
   else
      printf ("Erro, i eh maior que o numero de casas de N");
   return 0;
}
```

2.

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

float aproxima (float x, float err);
float potencia (float num, int exp);
int fatorial (int num);

int main ()
{
   float x, err;
   printf("Digite x e err: ");
   scanf("%f%f", &x, &err);
   printf("%f", aproxima(x, err));
   return 0;
}
```

```
15
16 float aproxima(float x, float err)
17 {
18    int i;
19    float resultado = 0, termo = err + 1;
20    for (i = 0; termo > err; i ++)
21    {
22       termo = potencia (x, 2*i + 1) / (3* fatorial (2*i));
23       /*termo = potencia (x, 2*i + 1) / (fatorial (2*i + 1)); --> Série de Taylor do seno para testar o
funcionamento do programa */
24       if (i % 2 == 0)
25          resultado += termo;
26       else
27          resultado -= termo;
28    }
29    return resultado;
30 }
31
32 float potencia (float num, int exp)
33 {
34    float resultado = 1;
35    if (exp == 0)
36       return 1;
37    for (; exp > 0; exp --){
38       resultado *= num;
39    }
40    return resultado;
41 }
42
43 int fatorial (int num)
44 {
45    int resultado;
46
47    if (num == 0)
48       return 1;
49
50    for (resultado = 1; num > 0; num --){
51       resultado *= num;
52    }
53    return resultado;
54 }
55
```

3.

```c
#include <stdio.h>

int calculaS (int n);

int main()
{
   int n;
   printf("Digite N: ");
   scanf("%d", &n);
    printf("%d", calculaS(n));
    return 0;
}

int calculaS(int n)
{
   int s;
   do{
      s = 0;
      while (n > 0){
         s += n%10;
         n/=10;
      }
      n = s;
   }while (s >= 10);
   return n;
}
```

4.

```c
#include <stdio.h>

void simplifica (int num, int den, int* rn, int* rd)
{
   int i;
   for (i = 2; i <= num && i <= den; i++){
      while (num%i == 0 && den%i == 0){
         num /= i;
         den /= i;
      }
   }
   *rn = num;
   *rd = den;
}

```

5.

```c
#include <stdio.h>
#define VETOR 512
void simplifica (int num, int den, int* rn, int* rd);
int main()
{
   int vetor[VETOR], den = 1, i, n, num = 0, rn, rd;
   printf("Digite n e entao os pares");
   scanf("%d", &n);
   for (i = 0; i < 2*n; i++){
      scanf("%d", &vetor[i]);
   }
   for (i = 1; i < 2*n; i += 2){
      if (den != vetor[i]){
         den *= vetor[i];
      }
   }
   for (i = 0; i < 2*n; i += 2){
      num += (den / vetor[i + 1])*vetor[i];
   }
   simplifica(num, den, &rn, &rd);
   printf("%d/%d", rn, rd);
   return 0;
}
```

6.

```c
#include <stdio.h>
int main()
{
   double soma, parte1, parte3, termo, epsilon;
   int denominador_parte2;

   printf("Digite o valor de epsilon: ");
   scanf("%lf", &epsilon);
   termo = soma = 0.5;
   parte1 = 1;
   denominador_parte2 = 1;
   parte3 = 0.5;

   while (termo >= epsilon)
   {
      denominador_parte2 += 2;
      parte1 *= (float)(denominador_parte2 - 2) / (denominador_parte2 - 1);
      parte3 /= 4.0;
```

```
19      termo = parte1 * (1.0 / denominador_parte2) * parte3;
20      soma += termo;
21    }
22    soma *= 6;
23    printf("Aproximacao do pi: %f \n", soma);
24    return 0;
25  }
```

7.

```
1  #include <stdio.h>
2
3  int collatz (double x, double eps);
4  int maxCollatz (double eps, int n, double u);
5  double absoluto (double x);
6
7  int main()
8  {
9     double x, epsilon, u, n;
10
11    printf("Digite o valor de x: ");
12    scanf("%lf", &x);
13    printf("Digite o valor de epsilon: ");
14    scanf("%lf", &epsilon);
15    printf("Digite o valor de n: ");
16    scanf("%lf", &n);
17    printf("Digite o valor de u: ");
18    scanf("%lf", &u);
19    printf("%d", maxCollatz(epsilon, n, u ));
20    return 0;
21  }
22
23  int collatz (double x, double eps)
24  {
25    int i;
26    for (i = 0; absoluto(x - 1) >= eps; i++){
27      if ((int)x % 2 == 0)
28        x /= 2;
29      else if ((3*(int)x + 1) % 4 == 0)
30        x = (3*x + 1)/4;
31      else
32        x = (3*x + 1)/2;
33    }
34    return i;
35  }
```

```
36
37 int maxCollatz (double eps, int n, double u)
38 {
39    double delta = (u - 1)/n;
40    double x, max = 0;
41    int i;
42    for (i = 0; i <= n; i++){
43       x = 1 + i * delta;
44       if (collatz(x, eps) > max)
45          max = collatz(x, eps);
46    }
47    return max;
48 }
49
50 double absoluto (double x)
51 {
52    if (x < 0){
53       printf("Batata"); /* Queria testar aqui a real necessidade de escrever essa função módulo */
54       return -x;
55    }
56    return x;
57 }
```

8.

```
 1 #define MAX 999
 2 #include <stdio.h>
 3
 4 int soma(int *res_m, int *res_e, int x_m, int x_e, int y_m, int y_e)
 5 {
 6    int precisao = 1, m_aux, e_aux;
 7    /* Guarda em x_m, x_e o operando de menor expoente*/
 8    if (y_e < x_e )
 9    { /* troca os valores de x_m, x_e e y_m, y_e */
10       m_aux = x_m;
11       e_aux = x_e;
12       x_m = y_m;
13       x_e = y_e;
14       y_m = m_aux;
15       y_e = e_aux;
16    }
17    /* Decrementa o expoente do maior numero ate que ele se iguale ao do menor,
18    mas sem estourar a mantissa do maior numero */
19    while (y_e > x_e && y_m*10 <= MAX && y_m * 10 >= -MAX){
20          y_e --;
```

```
21          y_m *= 10;
22      }
23      /* Se y_m chegou ao numero maximo de digitos e os expoentes ainda continuam diferentes,
24    entao e' preciso incrementar o expoente do menor numero cortando digitos de sua
25    mantissa. Nesse processo, o menor numero pode chegar a zero. */
26      while (y_e > x_e && x_m != 0){
27          x_e ++;
28          x_m /= 10;
29          precisao = 0;
30      }
31    /* o resultado da soma ficara armazenado temporariamente em y_m, y_e */
32      y_m += x_m;
33
34      if (y_m != 0) /* normaliza o resultado */
35
36        while (y_m % 10 == 0){
37          y_m /= 10;
38          y_e++;
39        }
40      if (y_m > MAX || y_m < -MAX){
41        /* |y_m| > MAX, entao e' preciso cortar 1 digito da mantissa */
42        y_m /= 10;
43        y_e ++;
44        precisao = 0;
45      }
46    /* armazena o resultado */
47      *res_m = y_m;
48      *res_e = y_e;
49      return precisao;
50    }
51
52    int main()
53    {
54      int x_m, x_e, y_m, y_e, res_m, res_e;
55      printf("Digite x_m, x_e, y_m e y_e: ");
56      scanf ("%d%d%d%d", &x_m, &x_e, &y_m, &y_e);
57
58      if(soma(&res_m, &res_e, x_m, x_e, y_m, y_e))
59        printf("%de%d + %de%d = %de%d", x_m, x_e, y_m, y_e, res_m, res_e);
60      else
61        printf("%de%d + %de%d = %de%d, com perda de precisao", x_m, x_e, y_m, y_e, res_m, res_e);
62
63      return 0;
64    }
```

9.

a) Incorreto: Após o laço, p está com 1 a mais do que deveria estar. Além disso, usa p1 e p2 quando deveria usar *p1 e *p2.
b) Correto
c) Correto
d) Incorreto: o laço para no i + 1-ésimo par de primos gêmeos, e p não é uma variável do tipo ponteiro, portanto não faz sentido usar *p.