



# SCC0221 - Introdução à Ciência de Computação I

---

**Prof.: Dr. Rudinei Goularte**

(rudinei@icmc.usp.br)

## Funções

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



# Sumário

---

- 1. Introdução
- 2. Forma Geral.
- 3. Protótipos de Funções.
- 4. Regras de Escopo.
- 5. Argumentos e Parâmetros.
- 6. Retorno de Valores.
- 7. O Comando Return.
- 8. Argumentos para main().



# 1. Introdução

---

- Trechos retirados do livro “A Book on C” de Kelley e Pohl – capítulo 5:
- “O coração da efetiva resolução de um problema é a decomposição do problema. Quebrar um problema em pedaços menores e mais gerenciáveis é crítico para escrever grandes programas.”



# 1. Introdução

---

- “Em C, as funções são usadas para implementar esse método *top-down* de programação”.
- “Um programa em C consiste de um ou mais arquivos contendo zero ou mais funções, uma delas sendo a *main()*.”



# 1. Introdução

---

- “A execução de um programa começa pela função *main()*, a qual pode chamar outras funções.”
  - Funções definidas pelo programador.
  - Funções de biblioteca: `printf()`, `scanf()`, `sqrt()`, ... .
- “Funções trabalham com variáveis do programa. Quando cada uma dessas variáveis está disponível em um local específico da função é determinado por regras de escopo.”<sub>5</sub>



# 1. Introdução

## Programa C

- **Diretivas ao Pré-Processador**
  - Includes
  - Macros
- **Definições de tipos**

- **Declarações Globais**
  - **Funções**
  - Variáveis

- **Definição das Funções**

- **Programa Principal**

```
int main (void)
{ /* begin */
} /* end */
```

↙ **/\*Isto é um  
comentário \*/**

→ **Programa  
mínimo em C**



## 2. Forma Geral

---

- *tipo nome\_da\_função (lista de parâmetros)*

{  
    *declarações e sentenças*  
}

- Tudo antes do “abre-chaves” compreende o **cabeçalho** da **definição** da função.
- Tudo entre as chaves compreende o **corpo** da **definição** da função.



## 2. Forma Geral

---

`char funcao (int x, char y)` → cabeçalho

`{`  
`char c;`  
`c = y + x;`  
`return (c);`  
`}` → corpo





## 2. Forma Geral

---

- *tipo nome\_da\_função (lista de parâmetros)*  
*{declarações sentenças}*
- **tipo:** é o tipo da função, i.e., especifica o tipo do valor que a função deve retornar (*return*).
  - Pode ser qualquer tipo válido.
  - Se a função não retorna valores, seu tipo é **void**.
  - Se o tipo não é especificado, tipo *default* é **int**.
  - Se necessário, o valor retornado será convertido para o tipo da função.



## 2. Forma Geral

---

```
char funcao (int x, char y) {  
    char c;  
    c = y + x;  
    return (c);  
}
```

```
void nada (int x, char y){  
    int c;  
    c = y + x;  
}
```

```
int f2 (int x, char y){  
    int c;  
    c = y + x;  
    return (c);  
}
```

```
int mult (void){  
    int x = 2;  
    float y = 3.34;  
    return (x*y);  
}
```



## 2. Forma Geral

---

- *tipo nome\_da\_função (lista de parâmetros) {declarações sentenças}*
- **nome\_da\_função**: pode ser qualquer identificador válido em C.



## 2. Forma Geral

---

- *tipo nome\_da\_função (lista de parâmetros)*  
*{declarações sentenças}*
- **lista parâmetros:** é uma lista de nomes de variáveis separadas por vírgula e seus tipos associados.
  - Cada variável **deve** ter seu tipo associado.
  - *f (tipo1 var1, tipo2 var2, ..., tipon varn);*
- Os parâmetros recebem os valores dos **argumentos** quando a função é chamada.
- Função sem parâmetros -> lista de parâmetros vazia.
  - Obs.: Mesmo assim os parênteses são necessários.



## 2. Forma Geral

---

```
int func (int a, char b, float c) {  
    char d;  
    d = a+ b+c;  
    return (d);  
}
```

```
int main (void){  
    int i, a=2;  
    char b='C';  
    float c=2.35;  
    ...  
    i = func(a, b, c);  
}
```

Os argumentos e os parâmetros devem ser em igual número. Além disso, os tipos devem estar na mesma ordem.



## 2. Forma Geral

---

- *tipo nome\_da\_função (lista de parâmetros) {declarações sentenças}*
- **declarações:**
  - toda variável declarada dentro do corpo de um função é dita ser uma **variável local** para essa função.
  - Uma Variável declarada externamente à uma função é dita ser **global**.



## 2. Forma Geral

---

```
#include <stdio.h>
int w;
int main (void){
    int i, j, w;                /* i é local p/ main*/
    ...
    w = (i * 3.5)/ i*i;
    j = funcao(w);              /* w é local*/
}
int funcao (int x) {
    char c; int y;              /*x, c e y são locais p/ funcao*/

    c = scanf("%c", &c);
    y = c + x + w; /*w é global*/
    return (y);
}
```



## 2. Forma Geral

---

- *tipo nome\_da\_função (lista de parâmetros) {declarações sentenças}*
- **sentenças**: além de declarações, o corpo de uma função pode conter qualquer sentença válida em C.
  - Sentenças podem ser expressões, comandos de controle de fluxo, atribuições, chamadas a funções, etc.





## 2. Forma Geral

---

```
int funcao (int x, char y, float z) {  
    char c;  
    x = y + z;  
    if (x > 10)  
        funcao2();  
    else{  
        scanf("%c", &c);  
        x = funcao3(c);  
    }  
    return (x);  
}
```

→ sentenças

# Programa C

- Diretivas ao Pré-Processador
  - Includes
  - Macros
- Definições de tipos

- Declarações Globais
  - **Funções**
  - Variáveis

- **Definição das Funções**

- Programa Principal

```
int main (void)
{ /* begin */
} /* end */
```

↙ **/\* Isto é um  
comentário \*/**

→ **Programa  
mínimo em C**

Exemplo de Programa.



## 3. Protótipos de Funções

---

- Até agora vimos exemplos de *definição de funções*.
- Funções deveriam ser declaradas antes de serem usadas.
- ANSI C definiu uma sintaxe para declaração de funções chamada *protótipo de função*.



## 3. Protótipos de Funções

---

- Qual o propósito?
  - Um protótipo de função informa ao compilador o número e o tipo dos argumentos que foram passados como parâmetros e o tipo do valor que a função deve retornar.
  - Facilita verificação de erros e inconsistências.



## 3. Protótipos de Funções

---

- Não é errado não usar protótipos.
  - Tecnicamente falando.
- O uso de protótipos constitui uma boa prática de programação.



# Exercício

---

- Declare protótipos de funções para:
  - Escrever uma mensagem na tela e retornar nada.
  - Somar dois inteiros e retornar o resultado.
  - Ler uma string e retornar o caracter mais freqüente.



## 4. Regras de Escopo

---

- Regra geral: identificadores de variáveis são acessíveis somente dentro do bloco no qual eles foram declarados.
- Função é um caso de bloco lógico.
- Em C, o código de uma função é privativo àquela função. Não pode ser acessado por nenhum comando em outra função, exceto por uma chamada à função.



## 4. Regras de Escopo

---

- E quando programadores usam o mesmo identificador em diferentes declarações?
- A qual variável o identificador se refere?
  - 1o. à variável declarada no mesmo bloco lógico, se existir.
  - 2o., à variável global.





## 4. Regras de Escopo

---

{

```
int a = 2;
```

```
printf("%d", a); /*imprime 2*/
```

```
{
```

```
    int a = 5;
```

```
    printf("%d", a); /*imprime 5*/
```

```
}
```

```
printf("%d", ++a);    /*imprime 3*/
```

```
}
```



## 4. Regras de Escopo

---

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

```
int funcao (int x);  
int funcao2(int y);  
int main (void){  
    int A=2, B, w=0;  
    ...  
    B = funcao(w);  
}
```

```
int funcao (int x) {  
    char c; int y, j;  
    y = c + x + A; /*erro*/  
    j = funcao2(y);  
    return (j);  
}
```

```
int funcao2(int y){  
    int x;  
    x = A + c; /*erro*/  
    return(x);  
}
```



## 4. Regras de Escopo

---

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

```
int funcao (int x, int A);  
int funcao2(int y, int A);  
int main (void){  
    int A=2, B;  
    ...  
    B = funcao(w, );  
}  
int funcao (int x, int A) {  
    char c; int y, j;  
    y = c + x + A;  
    j = funcao2(y, A);  
    return (j);  
}
```

```
int funcao2(int y, int A){  
    int x;  
    x = A + y; /*erro*/  
    return(x);  
}
```



## 5. Argumentos e Parâmetros

---

- Valores passados à função durante sua chamada.
- Uma função pode ou não ter argumentos.
- Se uma função usa argumentos, ela deve declarar variáveis que recebam os valores dos argumentos: os parâmetros.



## 5. Argumentos e Parâmetros

---

```
int func (int x, char y, float z) {  
    char c;  
    x = y + z;  
    return (x);  
}
```

```
int main (void){  
    int i, a=2;  
    char b='C';  
    float c=2.35;  
    ...  
    i = func(a, b, c);  
}
```

Os argumentos e os parâmetros devem ser em igual número. Além disso, os tipos devem estar na mesma ordem.



## 5. Argumentos e Parâmetros

---

- Os parâmetros são variáveis locais.
- Como qualquer variável local, são criadas na entrada da função e são destruídas na saída da mesma.
- Cabe ao programador assegurar a compatibilidade entre argumentos e parâmetros.
  - Nem sempre o compilador gera mensagem de erro!



## 5. Argumentos e Parâmetros

---

```
int func (int x, char y, float z) {  
    int c;  
    c = x + y + z;  
    return (x);  
}
```

```
int main (void){  
    int i, a=2;  
    char b='C';  
    float c=2.35;  
    ...  
    i = func(a, b, c); /* i = func(2, 'C', 2.35); i = func(a,2.2,'c'); */  
}
```



## 5.1 Passagem por Valor

---

- Em C, a passagem de parâmetros padrão (*default*) é por valor.
- Esse método copia o valor do argumento no parâmetro da função.
- Alterações no parâmetro, feitas dentro da função, não alteram o argumento.





## 5.1 Passagem por Valor

---

```
int main (void)
{
    int n = 3, sum;
    printf("%d \n",n); /*imprime 3*/

    sum = soma(n);

    printf("%d \n", n); /* imprime 3*/
    printf("%d", sum); /*imprime 6*/
    return 0;
}
```

```
int soma (int n)
{
    int sum=0;
    for ( ; n > 0; --n);
        sum += n;
    return(sum);
}
```



# Exercício

---

- O quê será impresso?

```
#include "stdio.h"
```

```
int z; /*não usem var global – isso é apenas um exemplo!*/
```

```
void f(int x){
```

```
    x = 2;
```

```
    z += x;
```

```
}
```

```
int main (void){
```

```
    z = 5;
```

```
    f(z);
```

```
    printf("z = %d\n", z);
```

```
    getch();
```

```
    return(0);
```

```
}
```



## 5.2 Passagem por Referência

---

- O endereço do argumento é copiado no parâmetro.
- Dentro da função, o endereço é usado para acessar o argumento real utilizado na chamada.
- As alterações feitas no parâmetro, dentro da função, afetam o argumento.



## 5.2 Passagem por Referência

---

- Exemplo na lousa.
  - Função swap, p/ trocar o valor de duas variáveis.
  - Primeiro com passagem por valor.
    - Não funciona!
  - Depois com passagem por referência.



## 5.2 Passagem por Referência

```
#include <stdio.h>
void swap (int *x, int *y);
void main (void)
{
    int i, j;
    i = 10;
    j = 20;
    swap (&i, &j);
    printf("%d %d", i, j);
}
```

```
void swap (int *x, int*y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```



# Exercício

---

- Escreva um programa que utilize uma função para ler três variáveis inteiras, a, b e c quaisquer e colocá-las em ordem crescente tal que o conteúdo de  $a < b < c$ . Não utilize variáveis globais. Os valores das variáveis devem ser impressos na função `main()`.



## 6. Retorno de Valores

---

- Como as funções retornam valores, elas podem ser usadas como operandos em expressões.
  - $x = \max(x, y) + 100;$
- Funções não podem receber uma atribuição:
  - $\max(x, y) = 100; /*errado*/$



## 6. Retorno de Valores

---

- Funções devolvem valores, mas não é necessário usar esse valor.
  - `printf ()` devolve o número de caracteres escritos.
    - `x = printf("caracteres escritos: ");`
    - `printf("%d", x);`





## 7. O Comando *return*

---

- Comando *return*
  - Provoca a saída imediata de uma função.
    - Controle retorna ao código chamador.
  - Pode ser usado para retornar valores.
    - `return;`
    - `return ++a;`
    - `return (a * b);`
    - `return a * b;`



## 7. O Comando return

---

- Se o comando return contém um expressão, o valor da expressão é passado ao código chamador.
- Se necessário, o valor é convertido para o tipo de retorno especificado na definição da função.



## 7. O Comando return

---

- Pode haver zero ou mais comandos *return* em uma função.
- Somente um é executado.
- Se não houver um comando return, o controle volta ao chamador quando “o fecha-chaves “}” da função é executado”.



## 7. O Comando return

---

- Todas as funções, exceto as do tipo void, retornam valores.
  - Esse valor deve estar especificado, explicitamente, no comando return.
  - Se não houver um comando return, o valor de retorno da função é indefinido.



## 7. O Comando return

---

```
int func (int a, int b){  
    if (a > b)  
        return(a);  
    else  
        return(b);
```

```
}  
int func (int a, int b){  
    return(a+b);  
    return(b*a);
```

```
}  
int func (int a, int b){  
    b= b * a;  
}
```



## 8. Argumentos para main()

---

- `main ()` recebe argumentos via linha de comando.
- Exemplo:
  - `$/> gcc nome_prog.c -o nome_prog`
- Argumentos de `main()`:
  - `argc`: inteiro que contém o número de argumentos da linha de comando.
  - `argv`: vetor de ponteiros para caracteres. Cada posição do vetor (string) é um argumento.
  - O primeiro argumento é o nome do programa.

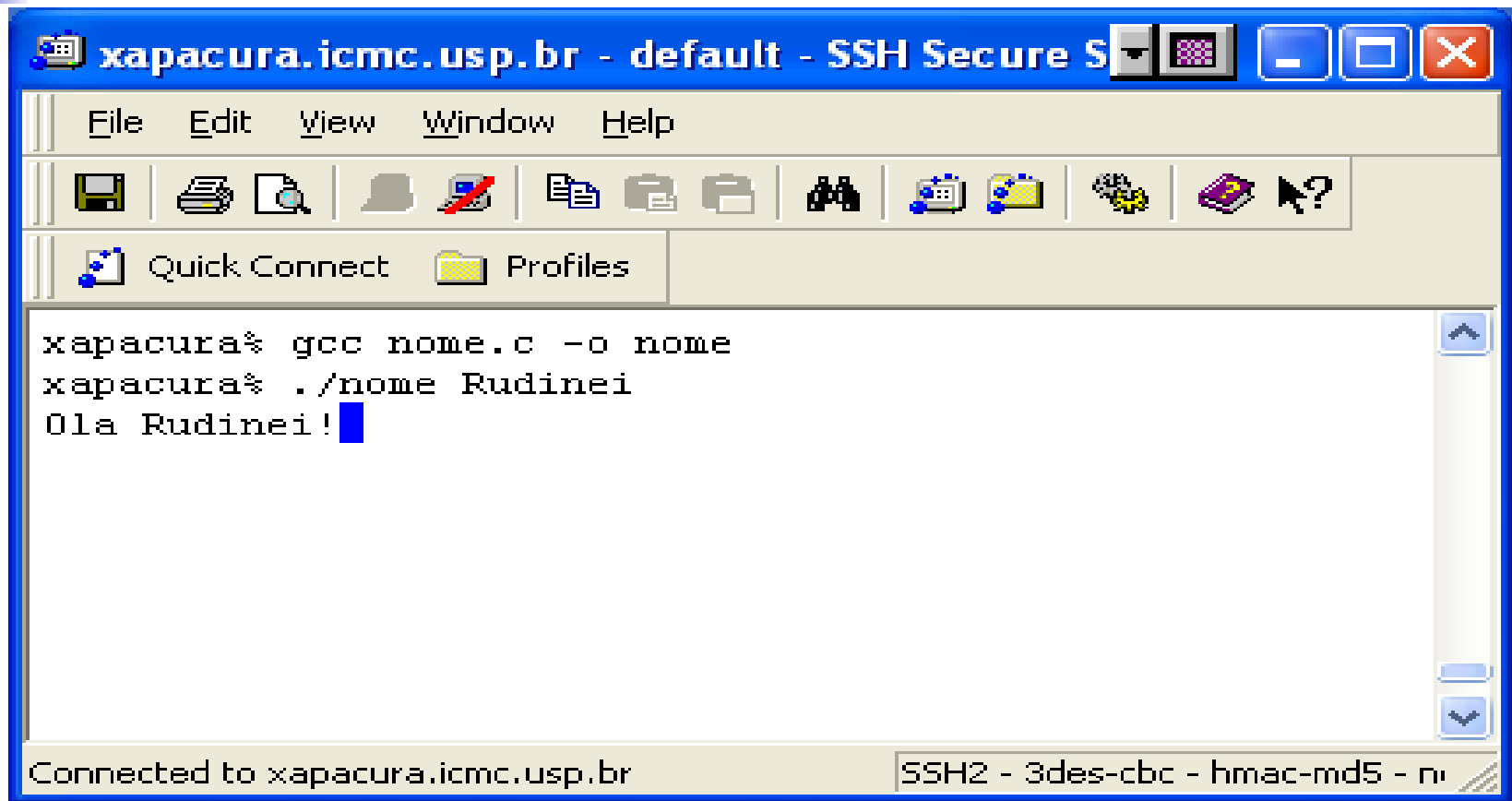


## 8. Argumentos para main()

---

```
/*arquivo nome.c*/
include <stdio.h>
int main(int argc, char *argv[])
{
    if (argc != 2){
        printf("Uso: nome <seu nome> <enter>" );
        exit(1);
    }
    printf("Olá %s!", argv[1]);
    return(0);
}
```

## 8. Argumentos para main()



The screenshot shows a terminal window titled "xapacura.icmc.usp.br - default - SSH Secure S". The window has a menu bar with "File", "Edit", "View", "Window", and "Help". Below the menu bar is a toolbar with various icons for file operations and system functions. The main area of the window displays the following text:

```
xapacura% gcc nome.c -o nome
xapacura% ./nome Rudinei
Ola Rudinei!
```

At the bottom of the window, a status bar indicates "Connected to xapacura.icmc.usp.br" and "SSH2 - 3des-cbc - hmac-md5 - n".

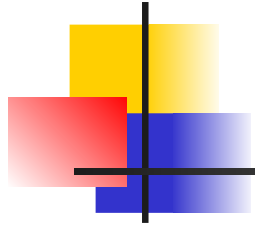




# Exercício

---

- Faça um programa calculadora para as quatro operações aritméticas básicas. Os argumentos (operandos e operadores) devem ser passados por linha de comando.
  - Exemplo: `#>./calc 3 + 5`



```
#include<stdio.h>
#include<stdlib.h>
int main (int argc, char *argv[]){
    float op1, op3;
    op1 = atof(argv[1]);
    op3 = atof(argv[3]);
    switch(argv[2][0]){
        case '+': printf("%f\n",op1+op3); break;
        case '-': printf("%f\n",op1-op3); break;
        case 'x': printf("%f\n",op1*op3); break;
        case '/': printf("%f\n",op1/op3); break;
    }

    return(0);
}
```



# O Quê main() Devolve?

---

- Devolve um inteiro para o processo chamador.
- Geralmente o chamador é o S.O.
- Se não está explícito, o valor devolvido é tecnicamente indefinido.
- Pode ser declarada como void.
  - Alguns compiladores geram uma advertência.



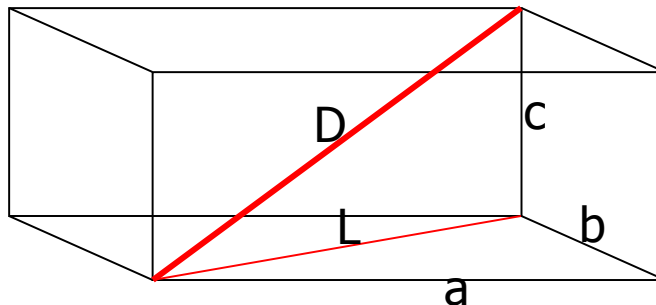
# Exercício

---

- Escreva um programa que utilize uma função para calcular a hipotenusa de um triângulo.

# Exercício

- Faça um programa que utilize a função hipotenusa, do exercício anterior, no cálculo da diagonal  $D$  de um paralelepípedo.
- Imprimir o valor da diagonal via programa principal.
- Somente os valores das arestas  $a$ ,  $b$  e  $c$  devem ser fornecidos pelo usuário.





# Passando Arrays como Argumentos

---

- *Arrays* são exceções à passagem *default* em C.
- *Arrays* como argumentos sempre passam seu endereço (o do primeiro elemento) ao parâmetro.



# Passando Arrays como Argumentos

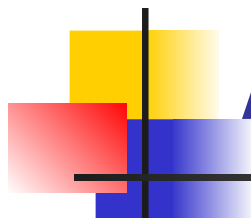
---

- Modos de passar um vetor (*array* unidimensional) como parâmetro:

```
int t[10]; ... display(t);
```

- `void display (int num[10]);`
- `void display (int num[]);`
- `void display (int *num);`

# Passando Arrays como Argumentos



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

```
void ToUpper (char *vet);
```

```
void ToUpper(char *vet){
    int i;

    for(i=0; i < strlen(vet)-1; i++){
        vet[i] = vet[i] - 32;
    }
}
```

```
int main (void){
```

```
    char s[80];
```

```
    printf("Digite uma frase: \n");
    fgets(s, 80, stdin);
    ToUpper(s);
    printf("\n%s", s);
```

```
    return(0);
```

```
}
```





# Passando Arrays como Argumentos

---

- Matrizes bidimensionais (*arrays* bidimensionais) como parâmetro.
  - Apenas a primeira dimensão pode ser omitida. Os colchetes, porém, são obrigatórios, assim como as demais dimensões.



# Passando Arrays como Argumentos

---

- Modos de passar uma matriz (*array* bidimensional) como parâmetro:

```
int t[10][5]; ... display(t);
```

- `void display (int num[10][5]);`
- `void display (int num[][5]);`
- `void display (int **num);`



# Passando Arrays como Argumentos

---

- Arrays de qualquer dimensão são sempre passados por referência.
- Arrays de duas ou mais dimensões:
  - Nos protótipos de funções, o compilador não espera que seja especificada a primeira dimensão (pode-se omitir). Contudo, as demais são obrigatórias.
    - `int f(int[][10][5]);`

# Passando Ponteiros como Argumentos

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

```
void f (int *p);
```

```
int main(void)
```

```
{
```

```
    int x=10, *p;
```

```
    p = &x;
```

```
    f(p);
```

```
    printf("x = %d\n",x);
```

```
    return(0);
```

```
}
```

```
void f (int *p)
```

```
{
```

```
    *p += 2;
```

```
}
```

**equivalente** →

```
int x =10;
```

```
f(&x);
```



# Retornando Ponteiros

---

- Para retornar um ponteiro, a função deve ser declarada como tendo tipo de retorno ponteiro:

```
char *match(char c, char *s){  
    while ((c != *s) && *s)  
        s++;  
    return(s);  
}
```



# Retornando Ponteiros

---

```
#include <stdio.h>
char *match(char c, char *s);
int main (void){
    char s[80], *p, ch;
    scanf(" %[^\n]s", s);
    scanf(" %c", &ch);
    p = match(ch, s);
    if (*p != '\0')
        printf ("%s ", p);
    else
        printf("não encontrei");
    return(0);
```

```
}
```



Fim da aula.

---