

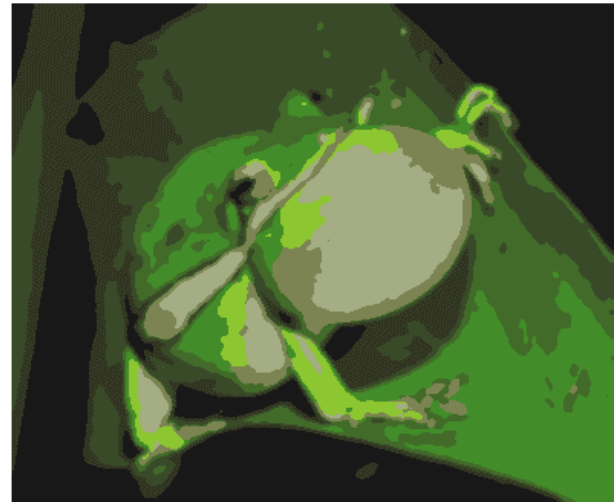
Programação 2

Revisão: Programação básica

Rivera

Algoritmo

- Conversão da imagem A para Imagem B



Algoritmo nível 1

- 1- Definir dados de entrada-saída
- 2- Eleição dos k-relevantes
- 3- Agrupar elementos em função da proximidade com k-relevantes
- 4- Verificar a dispersão de cada agrupamento.
Caso dispersão, efetuar novas agrupações
(ir para 3)
- 5- Mostrar resultados

Refinamentos

1: dados de entrada-saída

$E \leftarrow \text{Image (L, C)}$ // matriz de imagem

Se Abrir (E) error,

Cancelar....

$S \leftarrow \text{Image (L, C)}$ // imagem de saída

$T \leftarrow \text{Matriz int (L, C)}$ // mat de trabalho

Refinamentos

2: k-referentes

Ler (k) // $k > 1$

Definir

VKc [k] // formato RGB (centros)

VKi[k] // formato (int, int) - índices

VKm[K] // formato RGB (novos centros)

Para (h=1...k) // preenche vetores VK..

ik = random($\leq L$); jk = random ($\leq C$)

VKi[h] \leftarrow (ik, jk) // guarda índices

VKc[h] \leftarrow E[ik, jk] // copia Pixels

Refinamientos

3: Agrupamientos ()

Para ($i=1\dots L$)

Para ($j=1\dots C$)

$d_{\min} = 10000$ // máximo valor

para ($h=1\dots K$)

$d \leftarrow ||E[i, j] - VKc[h]||$

Se ($d < d_{\min}$)

$d_{\min} \leftarrow d$

$ik \leftarrow h$

$T[i, j] \leftarrow ik$ // copia índice de referencia p' T

Refinamentos

4: Dispersão

4.1: Calcular mediaAritmética dos pixels (em E) correspondentes a cada grupo

4.2:

Se ($|| \text{mediaAritmética} - \text{VKc} || > \text{VNulo}$)

$\text{VKc} \leftarrow \text{mediaAritmética}$

$\text{mediaAritmética} \leftarrow (0, \dots, 0)$

Ir para Agrupamento (3)

Senão

Gerar S como

$S[i,j] \leftarrow \text{VKc}[T[i,j]]$

Refinamientos

4.1: mediaAritmética

Para ($i=1\dots L$)

Para ($j=1\dots C$)

$ik \leftarrow T[i, j]$

$VKm[ik] += E[i, j]$

$VKt[ik] += 1;$

$d = 0.0$

Para ($ik = 1\dots K$)

$VKm[ik] /= VKt[ik]$

$d += ||VKc[ik] - VKm[ik]||$

Refinamentos

4.2: Decide

Se $(d > \text{limite})$

$VK_c \leftarrow VK_m$

$VK_m \leftarrow (0, \dots, 0)$

$VK_t \leftarrow (0, \dots, 0)$

Ir para Agrupaamento (3)

Senão

$S[., .] = VK_c[T[., .]]$

Fsi

Implementação

- Implementar em C o algoritmos analisados
- Revisão de C

Variáveis e Constantes

- Tipos básicos na linguagem C:

Tipo	Tamanho	Menor valor	Maior valor
char	1 byte	-128	+127
unsigned char	1 byte	0	+255
short int (short)	2 byte	-32.768	+32.767
unsigned short int	2 byte	0	+65.535
int (*)	4 byte	-2.147.483.648	+2.147.483.647
long int (long)	4 byte	-2.147.483.648	+2.147.483.647
unsigned long int	4 byte	0	+4.294.967.295
float	4 byte	-10^{38}	$+10^{38}$
double	8 byte	-10^{308}	$+10^{308}$

(*) depende da máquina, sendo 4 bytes para arquiteturas de 32 bits

Variáveis e Constantes

- Constante:

- ♦ armazenado na memória (tipo – sintaxe)

```
123          /* constante inteira do tipo "int" */
12.45        /* constante real do tipo "double" */
1245e-2      /* constante real do tipo "double" */
12.45F       /* constante real do tipo "float" */
```

- Variável:

- ♦ Declarada

```
int  a;      /* declara uma variável do tipo int */
int  b;      /* declara uma variável do tipo int */
float c;     /* declara uma variável do tipo float */
int  d, e;   /* declara duas variáveis do tipo int */
```

Variáveis e Constantes

- Declaração de variável:

```
int a;          /* declara uma variável do tipo int */  
a = 4.3;        /* a armazenará o valor 4 */
```

```
int a = 5, b = 10;
```

```
float c = 5.3;
```

```
int a, b, c;    // declara e inicializa duas variáveis do tipo
```

```
a = 2;
```

```
c = a + b;      /* ERRO: b contém "lixo" */
```

Operadores e Expressões

- Operadores:
 - ♦ aritméticos: $+$, $-$, $*$, $/$, $\%$
 - ♦ atribuição: $=$, $+=$, $-=$, $*=$, $/=$, $\%=$
 - ♦ incremento e decremento: $++$, $--$
 - ♦ relacionais e lógicos: $<$, $<=$, $==$, $>=$, $>$, $!=$
 - ♦ outros

Operadores e Expressões

- Operadores aritméticos ($+$, $-$, $*$, $/$, $\%$):
 - ♦ operações são feitas na precisão dos operandos

```
int a
double b, c;
a = 3.5;          /* a recebe o valor 3 */
b = a / 2.0;      /* b recebe o valor 1.5 */
c = 1/3 + b;      /* 1/3 retorna 0 */
                  /* c recebe o valor de b */
```

```
x % 2             /* o resultado será 0, se x for par */
a + b * c / d     /* é equivalente a (a + ((b * c) / d)) */
```

Operadores e Expressões

- Operadores de incremento e decremento (++ , --)
 - ♦ incrementa ou decrementa de uma unidade o valor de uma variável
 - os operadores não se aplicam a expressões
 - o incremento pode ser antes ou depois da variável ser utilizada

`n++` incrementa `n` de uma unidade, depois de ser usado

`++n` incrementa `n` de uma unidade, antes de ser usado

```
n = 5;  
x = n++;      /* x recebe 5; n é incrementada para 6 */  
x = ++n;      /* n é incrementada para 6; x recebe 6 */  
a = 3;  
b = a++ * 2;  /* b termina com o valor 6 e a com o valor 4 */
```


Operadores e Expressões

- Operadores relacionais (<, <=, ==, ==>, >, !=)
 - ♦ o resultado será 0 ou 1 (não há valores booleanos em C)

```
int a, b;  
int c = 23;  
int d = c + 4;
```

```
c < 20           retorna 0
```

```
d > c           retorna 1
```

Operadores e Expressões

- Operadores lógicos (`&&` , `||` , `!`)
 - ♦ a avaliação é da esquerda para a direita
 - ♦ a avaliação pára quando o resultado pode ser conhecido

```
int a, b;  
int c = 23;  
int d = c + 4;  
  
a = (c < 20) || (d > c);      /* retorna 1 */  
                               /* as duas sub-expressões são avaliadas */  
b = (c < 20) && (d > c);      /* retorna 0 */  
                               /* apenas a primeira sub-expressão é avaliada */
```

Operadores e Expressões

- sizeof :
 - ♦ retorna o número de bytes ocupados por um tipo

```
int a = sizeof(float)      /* armazena 4 em a */
```

Operadores e Expressões

- conversão de tipo :
 - ♦ conversão de tipo é automática na avaliação de uma expressão
 - ♦ conversão de tipo pode ser requisita explicitamente

```
float f;           /* valor 3 é convertido automaticamente para "float" */  
float f = 3;       /* ou seja, passa a valer 3.0F, antes de ser atribuído a f */  
int g, h;          /* 3.5 é convertido (e arredondado) para "int" */  
g = (int) 3.5;     /* antes de ser atribuído à variável g */  
h = (int) 3.5 % 2   /* e antes de aplicar o operador módulo "%" */
```

Entrada e Saída

- `#include <stdio.h>`
- Função “printf”:
 - ♦ possibilita a saída de valores segundo um determinado formato

```
printf (formato, lista de constantes/variáveis/expressões... );
```

```
printf ( “ %d %g ”, 33, 5.3 );
```

tem como resultado a impressão da linha:

33 5.3

```
printf ( "Inteiro = %d Real = %g ", 33, 5.3);
```

com saída:

Inteiro = 33 Real = 5.3

Entrada e Saída

- Especificação de formato:

`%c` *especifica um char*

`%d` *especifica um int*

`%u` *especifica um unsigned int*

`%f` *especifica um double (ou float)*

`%e` *especifica um double (ou float) no formato científico*

`%g` *especifica um double (ou float) no formato mais apropriado*

`%s` *especifica uma cadeia de caracteres*

- Escape

`\n` *nova linha*

`\t` *tabulação*

`\r` *retrocesso*

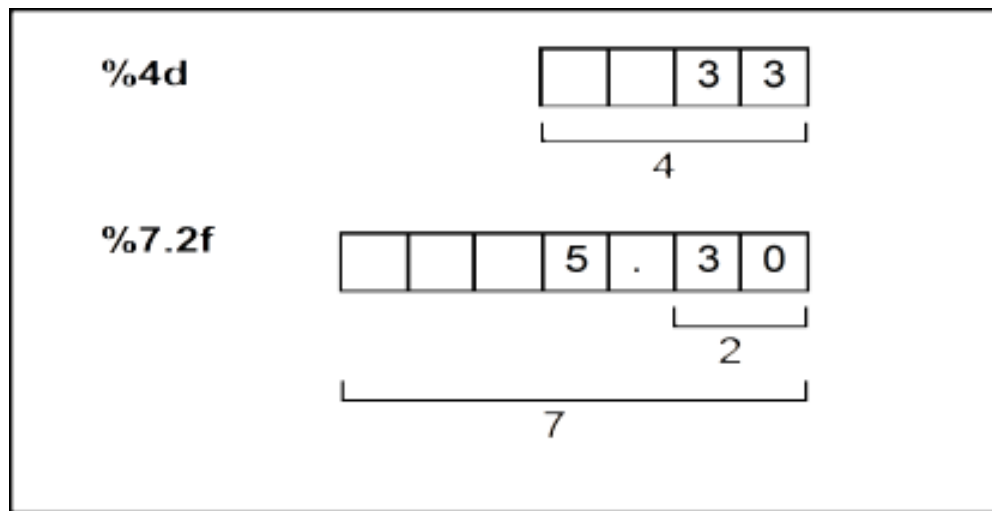
`\"` *caractere "*

`\\` *caractere *

`%%` *caractere %*

Entrada e Saída

Especificação de tamanho de campo:



Entradas e saídas

- Função “scanf”:
 - ♦ captura valores fornecidos via teclado

```
int n, m;
```

```
scanf ( “%d %d”, &n, &m);
```

```
scanf (“%d:%d”, &n, &m)
```

valor inteiro digitado pelo usuário é armazenado na variável n

Especificação de formato:

%c	especifica um char
%d	especifica um int
%u	especifica um unsigned int
%f,%e,%g	especificam um float
%lf, %le, %lg	especificam um double
%s	especifica uma cadeia de caracteres

Codificar

Tomada de decisão

```
if ( expr )    // se expr != 0
    { bloco de comandos 1 }
else          // se expr = 0
    { bloco de comandos 2 }
```

ou

```
if ( expr )
    { bloco de comandos }
```

```
int main (void)
{
    float nota ;
    printf ("Digite sua nota: ");
    scanf ("%f", &nota);
    if (nota >= 7 ) {
        printf (" Boa nota, parabens! \n" );
    }
    else {
        printf (" Voce precisa melhorar. \n");
    }
    return 0;
}
```

Tomada de decisão

```
/* temperatura (versao 1 - ???) */
#include <stdio.h>
int main (void)
{
    int temp;
    printf("Digite a temperatura: ");
    scanf("%d", &temp);
    if (temp < 30)
        if (temp > 20)
            printf(" Temperatura normal \n");
        else
            printf(" Temperatura quente \n");
    return 0;
}
```

```
/* temperatura (versao 2 - ???) */
#include <stdio.h>
int main (void)
{
    int temp;
    printf ( "Digite a temperatura: " );
    scanf ( "%d", &temp );
    if ( temp < 30 ) {
        if ( temp > 20 )
            printf ( " Temperatura normal \n" );
        }
    else
        printf ( " Temperatura quente \n" );
    return 0;
}
```

Tomada de decisão

- Operador condicional:

- ♦ formato geral:

condicao ? exp1 : exp2;

Ex.

maior = a > b ? a : b;

Equivale a

if (a > b)

 maior = a

else

 maior = b;

```
int valMaior( n, *v)
{
    int maior;
    maior = v[0];
    for (i=1; i<n; i++)
        maior = maior > v[i] ? maior : v[i];
    return (maior);
}
```

```
#define MAIOR (a, b) (a > b ? a : b)
int valMaior( n, *v)
{
    int maior;
    maior = v[0];
    for (i=1; i<n; i++)
        maior = MAIOR ( maior, v[i] );
    return (maior);
}
```

Construções com laços

- Comando “while”

```
while ( cond )  
{  
    bloco de comandos  
}
```

- Comando “do-while”

```
do  
{  
    bloco de comandos  
} while ( cond )
```

- Comando “for”

```
for ( explni; cond; explncre )  
{  
    bloco de comandos  
}
```

Equivalente a

```
explni;  
while (cond)  
{  
    bloco de comandos  
    explncre  
}
```

Construções com laços

- Interrupção de laços - Comando “break”:
 - ♦ termina a execução do comando de laço

Exem. Dada um string de n dados de caracteres, procurar um caractere específico

```
Int procuraCaractere ( char *ss, char p )
{
    int i, n, s;
    n = strlen ( ss );
    s = -1;
    for ( i=0; i < n; i++ )
    {
        if (ss[i] == p ){
            s = i;
            break;
        }
    }
    return (s);
}
```

Funções

- Comando para definição de função:

```
tipo nome_função (parâmetros... )  
{  
    corpo da função  
    retorna valor  
}
```

```
void nome_função (parâmetros... )  
{  
    corpo da função  
}
```

```
#include <stdio.h>  
int fat (int n);  
int main (void) {  
    int r, n = 5;  
    r = fat ( n );  
    printf ("Fat (%d )= %d \n", n, r);  
    return 0;  
}
```

```

int defineDados (int vn [ ] )
{
    int i, a, n;
    printf (" \n Digite n: ");
    scanf ("%d", &n );
    for (i = 1; i <= n; i++)
    {
        printf (" Entre o val %d-o: ", i);
        scanf ("%d", &a);
        vn[i-1] = a;
    }
    return ( n );
}

```

// -----(programa principal)-----

```

void main (void)
{
    int n, vn[100];
    n = defineDados(vn);
    mostraDados(n, vn);
    ordenaPar(n, vn);
    mostraDados(n, vn);
}

```

```

Void defineDados (int *n, int vn [ ] )
{
    int i, a;
    printf (" \n Digite n: ");
    scanf ("%d", n );
    for (i = 1; i <= *n; i++)
    {
        printf (" Entre o val %d-o: ", i);
        scanf ("%d", &a);
        vn[i-1] = a;
    }
}

```

// -----(programa principal)-----

```

void main (void)
{
    int n, vn[100];
    defineDados(&n, vn);
    mostraDados(n, vn);
    ordenaPar(n, vn);
    mostraDados(n, vn);
}

```



```
/* programa fatorial de um número */
```

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

```
int fat (int n);
```

```
int main (void)
```

```
{    int n, r;  
    printf("Digite um número nao negativo:");  
    scanf("%d", &n);  
    fat(n);  
    return 0;  
}
```

```
/* função para calcular o valor do fatorial */
```

```
int fat (int n)
```

```
{    int i, f = 1;  
    for (i = 1; i <= n; i++)  
        f *= i;  
    printf ("Fatorial = %f", f);  
}
```

```
/* programa fatorial de um número */
```

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

```
/* função para calcular o valor do fatorial */
```

```
int fat (int n)
```

```
{    int i, f = 1;  
    for (i = 1; i <= n; i++)  
        f *= i;  
    printf ("Fatorial = %f", f);  
}
```

```
int main (void)
```

```
{    int n, r;  
    printf("Digite um número nao negativo:");  
    scanf("%d", &n);  
    fat(n);  
    return 0;  
}
```

Exercícios

- 1) Dado um vetor v de n números inteiros. Definir um algoritmo (programa) para recolocar os elementos de v de forma que o primeiro número ($v[0]$) apareça em alguma posição j do vetor de forma que $v[0] \dots v[j-1] < v[j] \leq v[j+1] \dots v[n-1]$
- 2) Dados dois vetores va e vb de números inteiros ordenados em forma crescente de n e m elementos respectivamente. Definir um terceiro vetor vc ordenado, de forma eficiente, combinando elementos de va e vb .
- 3) Dado um vetor v de n números inteiros ordenados em forma crescente. Deseja-se buscar o número de ocorrências de um número a usando busca binária e sequencia parcial.
- 4) Usando o algoritmo em (1) escrever um algoritmo para a ordenação dos vetores parciais $v[0] \dots v[j-1]$ e $v[j+1] \dots v[n-1]$.

Pilha de Execução

Pilha: comunicação entre funções

c	'x'	112 - variável c no endereço 112 com valor 'x'
b	43.5	108 - variável b no endereço 108 com valor 43.5
a	7	104 - variável a no endereço 104 com valor 7

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */  
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )  
{   int n = 5;  
    int r;  
    r = fat ( n );  
    printf ("Fatorial de %d = %d \n", n, r);  
    return 0;  
}
```

```
int fat ( int n)  
{   int f = 1;  
    while (n != 0)  
    {  
        f *= n;  
        n--;  
    }  
    return f;  
}
```

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */  
#include <stdio.h>
```

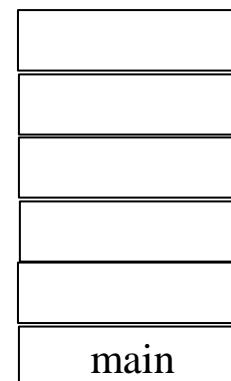
```
int fat ( int n );
```



```
Int main ( void )  
{   int n = 5;  
    int r;  
    r = fat ( n );  
    printf ("Fatorial de %d = %d \n", n, r);  
    return 0;  
}
```

```
int fat ( int n)  
{   int f = 1;  
    while (n != 0)  
    {  
        f *= n;  
        n--;  
    }  
    return f;  
}
```

1: Início do programa: pilha vazia



```
/* programa que lê um numero e imprime seu fatorial (versão 3) */  
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
```

```
{   int n = 5;  
    int r;  
    r = fat ( n );  
    printf ("Fatorial de %d = %d \n", n, r);  
    return 0;  
}
```

```
int fat ( int n)  
{   int f = 1;  
    while (n != 0)  
    {  
        f *= n;  
        n--;  
    }  
    return f;  
}
```


2: Declaração das variáveis: n, r

r	-
n	5
>	main

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
{   int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}
```



```
int fat ( int n)
{   int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```

3: chamada da função: cópia param.

n	5
>f	fat
r	-
n	5
>m	main

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
{
    int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

```
int fat ( int n)
{
    int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```



4: Declara variável local: f

f	1.0
n	5
>f	fat
r	-
n	5
>m	main


```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
{
    int n = 5;
    int r;
    r = fat ( n );
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

```
int fat ( int n)
{
    int f = 1;
    while (n != 0)
    {
        f *= n;
        n--;
    }
    return f;
}
```



5: Final do laço

f	120.0	
n	0	
>f	fat	
r	-	108
n	5	104
>m	main	100

```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
```

```
int fat ( int n );
```

```
Int main ( void )
```

```
{   int n = 5;
```

```
    int r;
```

```
    r = fat ( n );
```

```
    printf ("Fatorial de %d = %d \n", n, r);
    return 0;
```

```
}
```

```
int fat ( int n)
```

```
{   int f = 1;
```

```
    while (n != 0)
```

```
    {
```

```
        f *= n;
```

```
        n--;
```

```
    }
```

```
    return f;
```

```
}
```

6: Retorno da função: desempilha

r	120.0
n	5
>m	main

Funções Recursivas

- Tipos de recursão
 - ◆ Direta
 - Função A chama a ela própria
 - ◆ Indireta
 - Função A chama função B que, por sua vez, chama A
- Comportamento
 - ◆ Uso de ambiente local para cada chamada
 - ◆ Independência das variáveis locais em cada chamada
 - Como chamadas entre funções diferentes

Funções Recursivas

Exemplo: definição recursiva de fatorial

```
/* Função recursiva para cálculo do fatorial */  
int fat ( int n )  
{  
    if ( n == 0 )  
        return 1;  
    else  
        return n * fat ( n - 1 );  
}
```

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

/* Função recursiva para cálculo do fatorial */

```
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	-
n	5
>f	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

/* Função recursiva para cálculo do fatorial */

```
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



	1
	0
	fat(0)
	-
	1
	fat(1)
	-
	2
	fat(2)
	-
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```

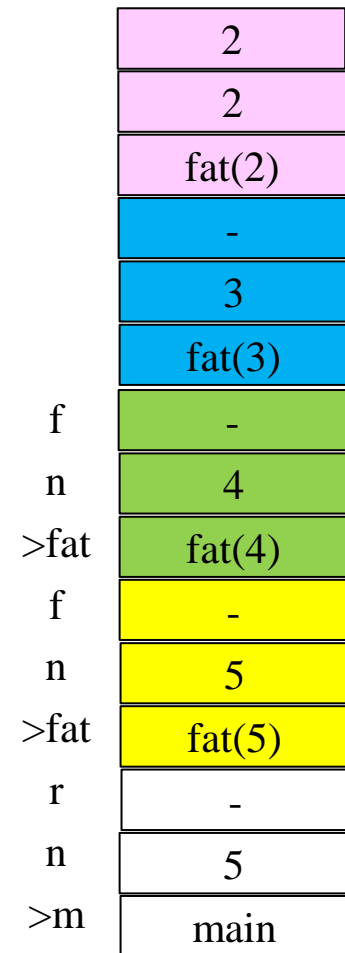



	1
	1
	fat(1)
	-
	2
	fat(2)
	-
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}

/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

/* Função recursiva para cálculo do fatorial */

```
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



	6
	3
	fat(3)
f	-
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

/* Função recursiva para cálculo do fatorial */

```
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	24
n	4
>fat	fat(4)
f	-
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}
```

/* Função recursiva para cálculo do fatorial */

```
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```



f	120
n	5
>fat	fat(5)
r	-
n	5
>m	main

Funções Recursivas

```
#include <stdio.h>
int fat (int n);
int main (void)
{   int n = 5;
    int r;
    r = fat ( n );
    printf("Fatorial de %d = %d \n", n, r);
    return 0;
}
```



```
/* Função recursiva para cálculo do fatorial */
int fat (int n)
{
    int f;
    if (n==0)
        f=1;
    else
        f= n*fat(n-1);
    return f;
}
```

r	120
n	5
>m	main

Variáveis Globais

- Variável global
 - ◆ Declarada fora do corpo das funções
 - Visível por todas as funções subsequentes
 - ◆ Não é armazenada na pilha de execução
 - Não deixa de existir quando a execução de uma função termina
 - Existe enquanto o programa estiver executando
 - ◆ Utilização de variáveis globais
 - Deve ser feito com critério
 - Pode-se criar um alto grau de interdependência entre as funções
 - Dificulta o entendimento e o reuso do código

Variáveis Globais

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

```
int s, p; /* variáveis globais */
```

```
void somaprod (int a, int b)
```

```
{
```

```
    s = a + b;
```

```
    p = a * b;
```

```
}
```

```
int main (void)
```

```
{
```

```
    int x, y;
```

```
    scanf("%d %d", &x, &y);
```

```
    somaprod(x,y);
```

```
    printf("Soma = %d produto = %d\n", s, p);
```

```
    return 0;
```

```
}
```

Variáveis Estáticas

- Variável estática
 - ♦ Declarada no corpo de uma função
 - Visível apenas dentro da função
 - ♦ Não é armazenada na pilha de execução
 - Armazenada em uma área de memória estática
 - Continua existindo antes e depois de a função ser executada
 - ♦ Utilização de variáveis estáticas
 - Quando for necessário recuperar o valor de uma variável atribuída na última vez que a função foi executada

Variáveis Estáticas

Exemplo

```
void imprime ( float a )  
{  
    static int n = 1;  
    printf ( " %f ", a);  
    if ( ( n % 5 ) == 0 ) printf( " \n " );  
    n++;  
}
```

Exemplo: ordenação de número

1. Definir n número nas tabelas
2. Ordenar os n números
 - Troca par (pivô, a)
 - Troca par consecutivo
 - Outros métodos
3. Mostrar resultados

..... **Algoritmo no quadro**

Ordenar números digitados

```
// ----( programa principal )----  
void main (void)  
{  
    int n, vn[100];  
    n = defineDados(vn);  
    mostraDados(n, vn);  
    ordenaPar(n, vn);  
    mostraDados(n, vn);  
}
```

```
int defineDados (int vn [ ] )  
{  
    int i, a, n;  
    printf (" \n Digite n: ");  
    scanf ("%d", &n );  
    for (i = 1; i <= n; i++)  
    {  
        printf (" Entre o val %d-o: ", i);  
        scanf ("%d", &a);  
        vn[i-1] = a;  
    }  
    return ( n );  
}
```

```
void mostraDados (int n, int vn[ ] )  
{  
    int i;  
    printf (" \n ----( %d dados do vetor )----- \n", n);  
    for (i=0; i<n; i++)  
        printf ("%d ", vn[i]);  
  
    printf ("\n ----( xxxxxx )---- \n");  
}
```

```
void ordenaPar ( int n, int vn [ ] )  
{  
    int troca, a, pv, i;  
    do {  
        troca = 0;  
        pv = vn [ 0 ];  
        for ( i = 1; i < n; i++ )  
        {  
            a = vn [ i ];  
            if (pv > a )  
            {  
                vn [ i - 1 ] = a;  
                vn [ i ] = pv;  
                troca++;  
            }  
            else  
                pv = a;  
        }  
    } while ( troca );  
}
```

Ordenar números de um arquivo

```
void ordenaPar ( int n, int vn [ ] )
{
    int troca, a, pv, i;
    do {
        troca = 0;
        pv = vn [ 0 ];
        for ( i = 1; i < n; i++ )
        {
            a = vn [ i ];
            if (pv > a )
            {
                vn [ i - 1 ] = a;
                vn [ i ] = pv;
                troca++;
            }
            else
                pv = a;
        }
    } while ( troca );
}
```

```
int defineDados (int vet [ ] )
{
    FILE *pointArq;
    int i, a, m;
    if ( ( pointArq = fopen ( nomArq, "rt" ) ) == NULL )
    {
        printf ( "Error em %s \n", nomArq );
        return 0;
    }
    fgets ( token, 40, pointArq );
    fscanf ( pointArq, "%d %s", &m, token);
    for ( i = 1; i <= m; i++ )
    {
        scanf (pointArq, "%d", &a);
        vn[i-1] = a;
    }
    fclose ( pointArq );
    return (m);
}
```

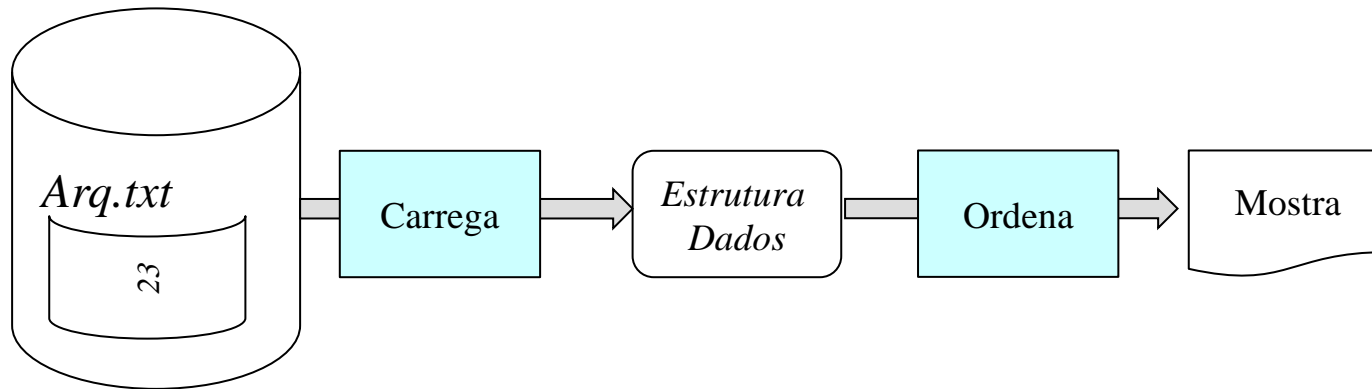
```
void mostraDados (int n, int vn [ ] )
{
    int i;
    printf ( " \n ----( %d dados do vetor )----- \n", n);
    for(i=0; i<n; i++)
        printf("%d ", vn[i]);

    printf("\n ----( xxxxxx )---- \n");
}
```

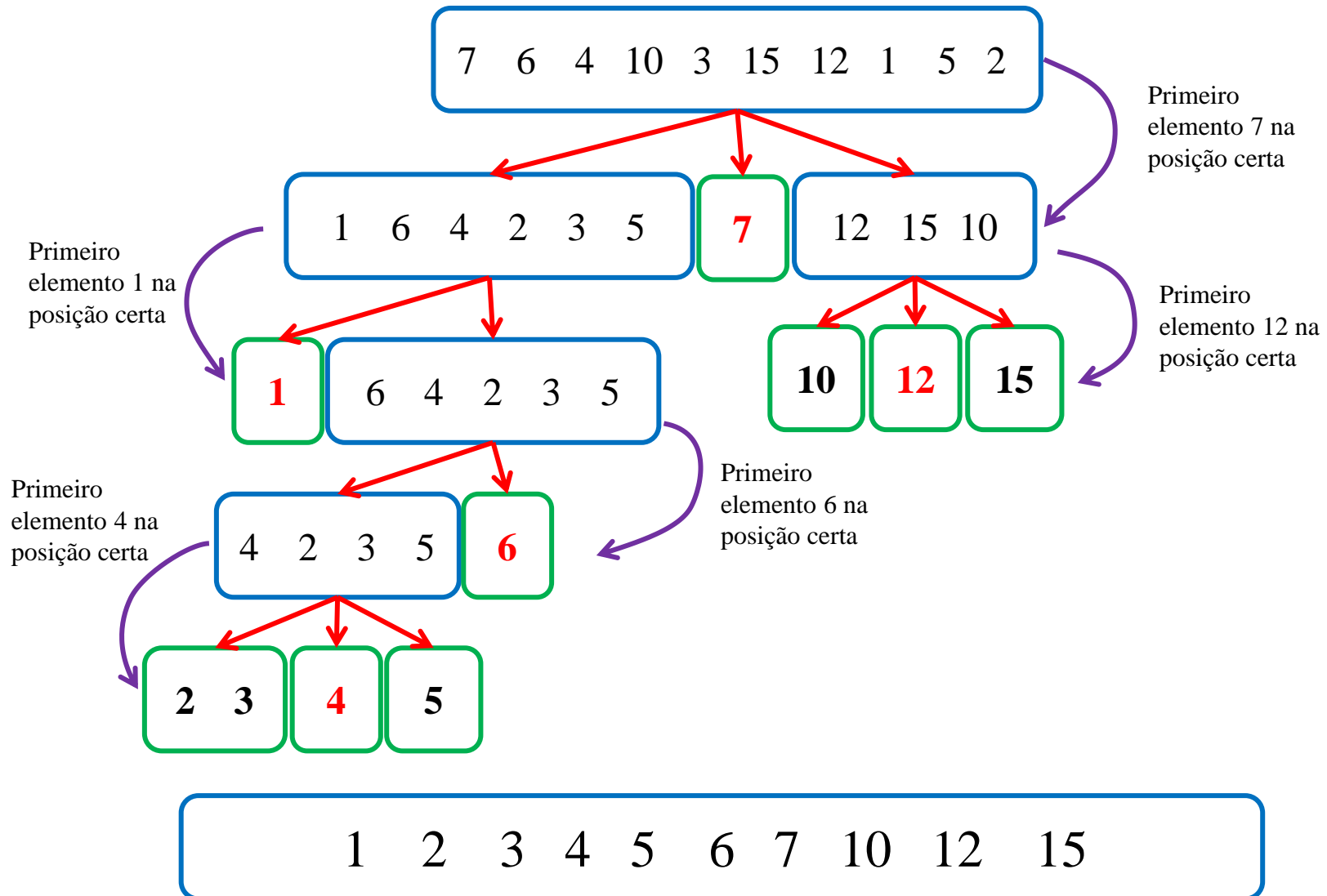
```
// -----( programa principal )-----

void main (void)
{
    int n, vn[100];
    n = defineDados("numOrdenar.txt", vn);
    mostraDados(n, vn);
    ordenaPar(n, vn);
    mostraDados(n, vn);
}
```

Ordenar dados de um arquivo



Estrutura binária (recursiva)



Algoritmo básico

Ordenar (ai, bi, v)

iniciar: a, b, pivô

colocar pivô na posição certa b

ordenar (a1, b-1, v)

ordenar (b+1, b1, v)

Fim.

Ordenar (ai, bi, v)

iniciar: a, b, pivô

Enquanto $a < b$, fazer

 caminhar $a++$ até obstáculo maior

 caminhar $b--$ até obstáculo menor

Se ($a < b$)

 trocar $v[a]$ com $v[b]$

Fenq.

colocar pivô na posição b // posicao certa

ordenar (a1, b-1, v)

ordenar (b+1, b1, v)

Fim.

Exercícios

- Projetar o algoritmo e implementar em C:
 1. Cálculo do valor de $PI = 4 * (1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + \dots)$
 2. Geração de n pontos (pares real) no plano, calcular centroide, e transladar origem dos ponto para esse centroide.