INF1007: Programação 2

0 – Revisão / Funções Recursivas

Tópicos Principais

- Variáveis e Constantes
- Operadores e Expressões
- Entrada e Saída
- Tomada de Decisão
- Construções com Laços
- Definição de Funções
- Pilha de Execução
- Funções Recursivas
- Variáves Globais X Variáveis Estáticas

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 bytes	-32.768	+32.767
`unsigned short int`	2 bytes	0	+65.535
`int (*)`	4 bytes	-2.147.483.648	+2.147.483.647
`long int (long)`	4 bytes	-2.147.483.648	+2.147.483.647
`unsigned long int`	4 bytes	0	+4.294.967.295
`float`	4 bytes	-10^{38}	$+10^{38}$
`double`	8 bytes	-10^{308}	$+10^{308}$

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

Variáveis e Constantes

- Valor Constante:
  - armazenado na memória
  - possui um tipo, indicado pela sintaxe da constante

```
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" */
```

Variáveis e Constantes

- Variável:
  - espaço de memória para armazenar um dado
  - não é uma variável no sentido matemático
  - possui um tipo e um nome
    - nome: identifica o espaço de memória
    - tipo: determina a natureza do dado

Variáveis e Constantes

- Declaração de variável:
  - variáveis devem ser explicitamente declaradas
  - variáveis podem ser declaradas em conjunto

```
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:
  - variáveis só armazenam valores do mesmo tipo com que foram declaradas

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

Variáveis e Constantes

- Variável com valor indefinido:
  - uma variável pode receber um valor quando é definida (inicializada), ou através de um operador de atribuição

```
int a = 5, b = 10;      /* declara e inicializa duas variáveis do tipo int */

float c = 5.3; /* declara e inicializa uma variável  do tipo float */
```

Variáveis e Constantes

- Variável com valor indefinido:
  - uma variável deve ter um valor definido quando é utilizada

```
int a, b, c; /* declara e inicializa duas variáveis do tipo int */
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
    - o operando com tipo de menor expressividade é convertido para o tipo do operando com tipo de maior expressividade
    - divisão entre inteiros trunca a parte fracionária

```
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 pois a operação será sobre inteiros */
                /* c recebe o valor de b */
```

Operadores e Expressões

- Operadores aritméticos (cont.):
  - o operador módulo, “%”, aplica-se a inteiros
  - precedência dos operadores: * , / , - , +

```
x % 2           /* o resultado será 0, se x for par;
                   caso contrário, será 1 */

a + b * c / d  é equivalente a       (a + ((b * c) / d))
```

Operadores e Expressões

- Operadores de atribuição :

  `( = , += , -= , *= , /= , %= )`

  – C trata uma atribuição como uma expressão

    - a ordem é da direita para a esquerda

  – C oferece uma notação compacta para atribuições em que a mesma variável aparece dos dois lados

var *op*= expr é equivalente a var = var *op* (expr)

```
i += 2;          é equivalente a    i = i + 2;
x *= y + 1;      é equivalente a    x =  x * (y +1);
```

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

- 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
```

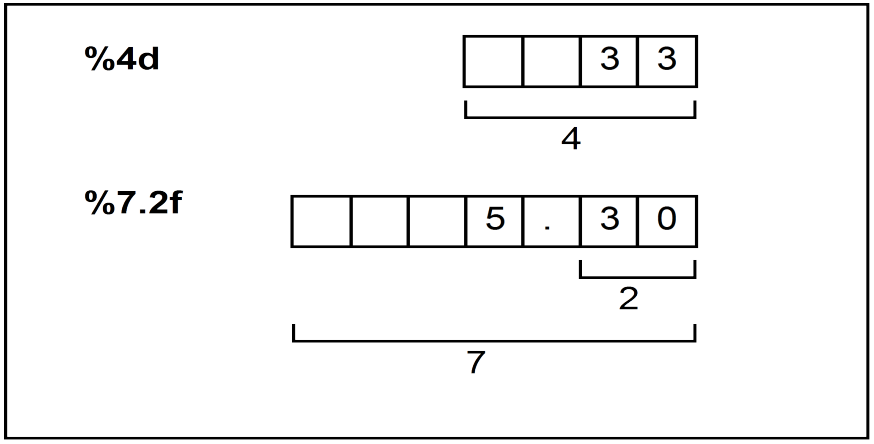
Entrada e Saída

- 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 (%f ou %e)*
`%s`	*especifica uma cadeia de caracteres*

Entrada e Saída

- Especificação de tamanho de campo:



Entrada e Saída

- Impressão de texto:

```
printf("Curso de Estruturas de Dados\n");

exibe na tela a mensagem:
Curso de Estruturas de Dados
```

Entrada e Saída

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

```
scanf (formato, lista de endereços das variáveis...);
```

```
int n;
scanf ("%d", &n);

valor inteiro digitado pelo usuário é armazenado na variável n
```

Entrada e Saída

- 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*

Entrada e Saída

- Função “scanf” (cont.):
  - caracteres diferentes dos especificadores no formato servem para cercar a entrada
  - espaço em branco dentro do formato faz com que sejam "pulados" eventuais brancos da entrada
  - %d, %f, %e e %g automaticamente pulam os brancos que precederem os valores numéricos a serem capturados

```
scanf ("%d:%d", &h, &m);

valores (inteiros) fornecidos devem ser separados pelo
caractere dois pontos (:)
```

Tomada de Decisão

- Comando “if”:
  - comando básico para codificar tomada de decisão
    - se expr for verdadeira (≠ 0), executa o bloco de comandos 1
    - se expr for falsa (= 0), executa o bloco de comandos 2

```
if ( expr )
{ bloco de comandos 1 }
else
{ bloco de comandos 2 }

ou

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

Exemplo

```
/* nota */
#include <stdio.h>

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;
}
```

Exemplo

```
/* nota */
#include <stdio.h>

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;
}
```

Bloco de comandos

```
{
 comando1;
 comando2;
 ...
 }
```

ou

```
 comando;
```

Tomada de Decisão

- Exemplo:
  - função para qualificar a temperatura:

se a temperatura for menor do que 20^{o}C, então está frio

se a temperatura estiver entre 20^{o}C e 30^{o}C, então está agradável

se a temperatura for maior do que 30^{o}C, então está quente

Tomada de Decisão

```
/* temperatura (versao 1 - incorreta) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);
   if (temp < 30)
      if (temp > 20)
         printf(" Temperatura agradável \n");
   else
      printf(" Temperatura quente \n");
   return 0;
}
```

Em C, um `else` está associado ao último `if` que não tiver seu próprio `else`.

Tomada de Decisão

```
/* temperatura (versao 1 - incorreta) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);
   if (temp < 30)
      if (temp > 20)
         printf(" Temperatura agradável \n");
   else
      printf(" Temperatura quente \n");
   return 0;
}
```

Tomada de Decisão

```
/* 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 agradável \n" );
   }
   else
      printf ( " Temperatura quente \n" );
   return 0;
}
```

```c
/* temperatura  (versao 3) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);

   if (temp < 10)
      printf("Temperatura muito fria \n");
   else if (temp < 20)
      printf(" Temperatura fria \n");
   else if (temp < 30)
      printf("Temperatura agradável \n");
   else
      printf("Temperatura quente \n");
   return 0;
}
```

```
/* temperatura  (versao 3) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);

   if (temp < 10)
      printf("Temperatura muito fria \n");
   else if (temp < 20)
       printf(" Temperatura fria \n");
   else if (temp < 30)
       printf("Temperatura agradável \n");
   else
       printf("Temperatura quente \n");
   return 0;
}
```

```
/* temperatura  (versao 3) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);

   if (temp < 10)
      printf("Temperatura muito fria \n");
      else if (temp < 20)
         printf(" Temperatura fria \n");
         else if (temp < 30)
             printf("Temperatura agradável \n");
             else
                printf("Temperatura quente \n");
   return 0;
}
```

```c
/* temperatura  (versao 3) */
#include <stdio.h>

int main (void)
{
   int temp;
   printf("Digite a temperatura: ");
   scanf("%d", &temp);

   if (temp < 10)
      printf("Temperatura muito fria \n");
      else if (temp < 20)
         printf(" Temperatura fria \n");
         else if (temp < 30)
             printf("Temperatura agradável \n");
             else
                printf("Temperatura quente \n");
   return 0;
}
```

Tomada de Decisão

- Estrutura de bloco:
  - declaração de variáveis:
    - só podem ocorrer no início do corpo da função ou de um bloco
    - (esta restrição não existe no C99)
  - escopo de uma variável:
    - uma variável declarada dentro de um bloco é válida no bloco
    - após o término do bloco, a variável deixa de existir

```
if ( n > 0 )
    { int i;  ... }
...           /* a variável i não existe neste ponto do programa */
```

Tomada de Decisão

- Operador condicional:
  - formato geral:
    - se a condição for verdadeira, a expressão1 é avaliada; caso contrário, a expressão2 é avaliada

```
condição ? expressão1 : expressão2;
```

  - exemplo:
    - comando

```
maximo = a > b ? a : b ;
```

    - comando "if" equivalente

```
if ( a > b )
   maximo = a;
else
   maximo = b;
```

Construções com laços

- Exemplo:
  - fatorial de um número inteiro não negativo:

$$n! = n \times (n-1) \times (n-2) \ldots 3 \times 2 \times 1$$

$$onde: 0! = 1$$

Construções com laços

- Exemplo:
  - definição recursiva da função *fatorial:* $N \rightarrow N$

fatorial(0) = 1

fatorial(n) = n x fatorial(n-1)

  - cálculo não recursivo de *fatorial(n)*
    - **comece com:**

```
k = 1
fatorial = 1
```

    - **faça enquanto** $k \leq n$

```
fatorial = fatorial * k
incremente k
```

Construções com laços

- Comando “while”:
  - enquanto *expr* for verdadeira, o bloco de comandos é executado
  - quando *expr* for falsa, o comando termina

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

```
/* Fatorial */
#include <stdio.h>
int main (void)
{
   int i;
   int n;
   long int f = 1;
   printf("Digite um numero inteiro nao negativo:");
   scanf("%d", &n);

   /* calcula fatorial */
   i = 1;
   while (i <= n)
   {
      f = f * i;         /* equivalente a "f *= i"  */
      i = i + 1;       /*   equivalente a "i++"    */
   }
   printf(" Fatorial = %d \n", f);
   return 0;
}
```

Construções com laços

- Comando “for”:
  - forma compacta para exprimir laços

```
for (expressão_inicial; expressão_booleana; expressão_de_incremento)
{
   bloco de comandos
}
```

  - equivalente a:

```
expressão_inicial;
while ( expressão_booleana )
{
   bloco de comandos
   ...
   expressão_de_incremento
}
```

```
/* Fatorial (versao 2) */
#include <stdio.h>

int main (void)
{
   int i;
   int n;
   int f = 1;

   printf("Digite um numero inteiro nao negativo:");
   scanf("%d", &n);

   /* calcula fatorial */
   for (i = 1; i <= n; i=i+1) {
      f = f * i;
   }
   printf(" Fatorial = %d \n", f);
   return 0;
}
```

```
/* Fatorial (versao 2) */
#include <stdio.h>

int main (void)
{
   int i;
   int n;
   int f = 1;

   printf("Digite um numero inteiro nao negativo:");
   scanf("%d", &n);

   /* calcula fatorial */
   for (i = 1; i <= n; i+1) {        /* o que acontece com este programa?          */
      f = f * i;
   }
   printf(" Fatorial = %d \n", f);
   return 0;
}
```

Construções com laços

- Comando “do-while”:
  - teste de encerramento é avaliado no final

```
do
{
   bloco de comandos
} while (expr);
```

```c
/* Fatorial (versao 3) */
#include <stdio.h>
int main (void)
{
   int i;
   int n;
   int f = 1;
   /* requisita valor até um número não negativo ser informado */
   do
   {
      printf("Digite um valor inteiro nao negativo:");
      scanf ("%d", &n);
   } while (n<0);
   /* calcula fatorial */
   for (i = 1; i <= n; i++)
      f *= i;
   printf(" Fatorial = %d\n", f);
   return 0;
}
```

```c
/* Fatorial (versao 4) */
#include <stdio.h>
int main (void)
{
   int i;
   int n;
   int f = 1;
   /* O que faz este programa? */
   do {
          printf("Digite um valor inteiro nao negativo:");
          scanf ("%d", &n);
          /* calcula fatorial */
           for (i = 1; i <= n; i++)
                       f *= i;
          printf(" Fatorial = %d\n", f);
       } while (n>=0);
   return 0;
}
```

Construções com laços

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

```
#include <stdio.h>
int main (void)
{
   int i;
   for (i = 0; i < 10; i++) {
      if (i == 5)
         break;
      printf("%d  ", i);
   }
   printf("fim\n");
   return 0;
}
```

A saída deste programa, se executado, será: 0 1 2 3 4 fim

Construções com laços

- Interrupção de laços - Comando “continue”:
  - termina a iteração corrente e passa para a próxima

```
#include <stdio.h>

int main (void)
{
   int i;
   for (i = 0; i < 10; i++ ) {
      if (i == 5) continue;
      printf("%d  ", i);
   }
   printf("fim\n");
   return 0;
}
```

gera a saída: 0 1 2 3 4 ~~5~~ 6 7 8 9 fim

Construções com laços

- Interrupção de laços - Comando “continue”:
  - deve-se ter cuidado para criar uma “iteração eterno”

```
/* INCORRETO */
#include <stdio.h>
int main (void)
{
   int i = 0;
   while (i < 10) {
      if (i == 5) continue;
      printf("%d  ", i);
      i++;
   }
   printf("fim\n");
   return 0;
}
```

cria “iteração eterna” pois i não será mais incrementado quando chegar a 5

Construções com laços

- Comando “switch”:
  - seleciona uma entre vários casos
    (“op_k” deve ser um inteiro ou caractere)

```
switch ( expr )
{
   case op1: bloco de comandos 1; break;
   case op2: bloco de comandos 2; break;
...
   default:  bloco de comandos default; break;
}
```

```
/* calculadora de quatro operações */
#include <stdio.h>
int main (void)
{
   float num1, num2;
   char op;
   printf("Digite: numero op numero\n");
   scanf ("%f %c %f", &num1, &op, &num2);
   switch (op)
   {
      case '+':  printf(" = %f\n", num1+num2); break;
      case '-':  printf(" = %f\n", num1-num2);  break;
      case '*':  printf(" = %f\n", num1*num2);  break;
      case '/':  printf(" = %f\n", num1/num2);  break;
      default: printf("Operador invalido!\n");  break;
   }
   return 0;
}
```

Definição de Funções

- Comando para definição de função:

```
tipo_retornado  nome_da_função  ( lista de parâmetros... )
{
   corpo da função
}
```

```
/* programa que lê um número e imprime seu fatorial */

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

"protótipo" da função: deve ser incluído antes da função ser chamada

chamada da função

"main" retorna um inteiro:
0 : execução OK
≠ 0 : execução ¬OK

declaração da função: indica o tipo da saída e o tipo e nome das entradas

```
void fat (int n);   /* obs: existe ; no protótipo */


void fat(int n)     /* obs: não existe ; na declaração */
{
}
```

```c
/* programa que lê um número e imprime seu fatorial (versão
2) */
#include <stdio.h>
int fat (int n);
int main (void)
{  int n, r;
   printf("Digite um número nao negativo:");
   scanf("%d", &n);
   r = fat(n);
   printf("Fatorial = %d\n", r);
   return 0;
}

/* função para calcular o valor do fatorial
int fat (int n)
{  int i;
   int f = 1;
   for (i = 1; i <= n; i++)
      f *= i;
   return f;
}
```

"protótipo" da função: deve ser incluído antes da função ser chamada

chamada da função

"main" retorna um inteiro:
0 : execução OK
≠ 0 : execução ¬OK

declaração da função: indica o tipo da saída e o tipo e nome das entradas

retorna o valor da função

Pilha de Execução

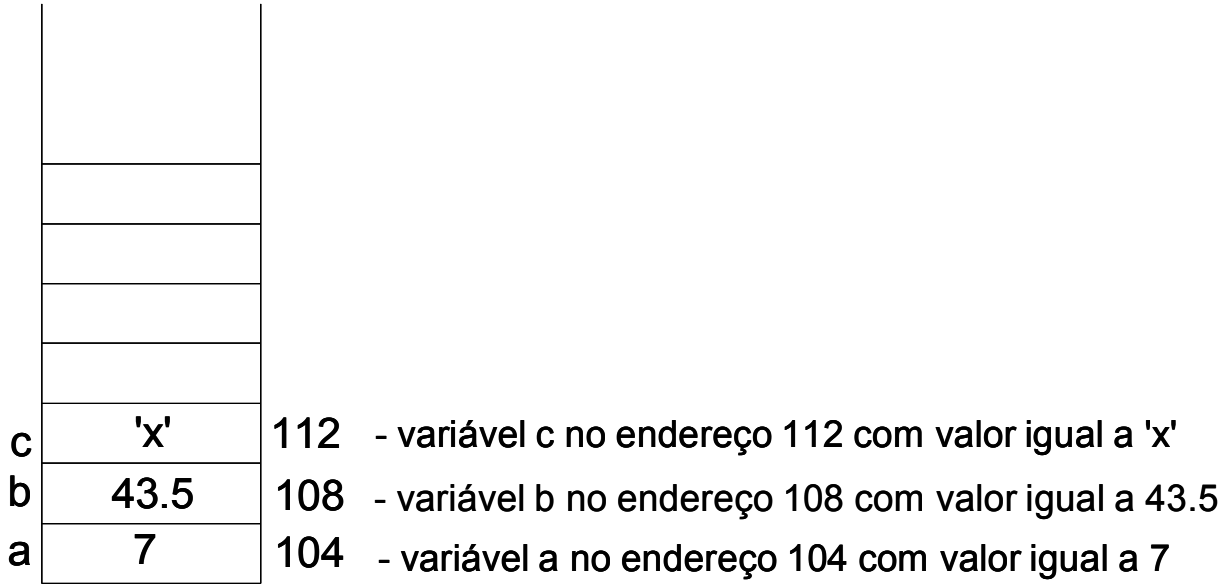
- Comunicação entre funções:
  - funções são independentes entre si
  - transferência de dados entre funções:
    - através dos parâmetros e do valor de retorno da função chamada
    - passagem de parâmetros é feita ***por valor***
  - variáveis locais a uma função:
    - definidas dentro do corpo da função (incluindo os parâmetros)
    - não existem fora da função
    - são criadas cada vez que a função é executada
    - deixam de existir quando a execução da função terminar

Pilha de Execução

- Comunicação entre funções (cont.):

Pergunta: Como implementar a comunicação entre funções?

Resposta: Através de uma pilha



Exemplo: Fatorial iterativo

```
/* 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;
}
```

declaração das variáveis `n` e `r`, locais à função `main`

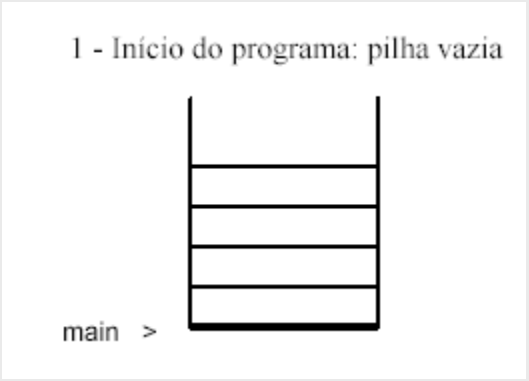
declaração das variáveis `n` e `f`, locais à função `fat`

alteração no valor de `n` em `fat` não altera o valor de `n` em `main`

simulação da chamada `fat(5)`:
a variável `n` possui valor 0 ao final da execução de `fat`, mas o valor de `n` no programa principal ainda será 5

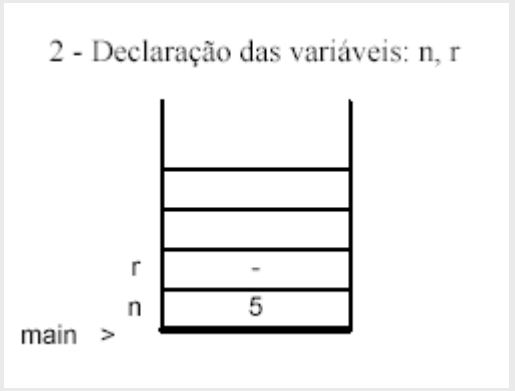
Exemplo: Início do programa

```
/* 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;
}
```



Exemplo: Declaração de `n` e `r` na `main()`

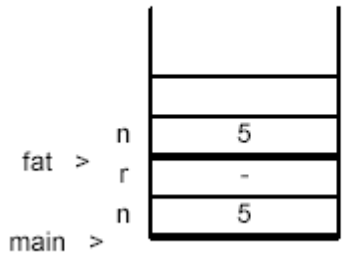
```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
int fat (int n);
int main (void)
{  int n = 5;
   int r;
   r = fat ( n );
   printf("Fatorial de %d = %d \n", n, r);
   return 0;
}
int fat (int n)
{  int f = 1;
   while (n != 0) {
      f *= n;
      n--;
   }
   return f;
}
```



Exemplo: Declaração de `n` na `fat(int n)`

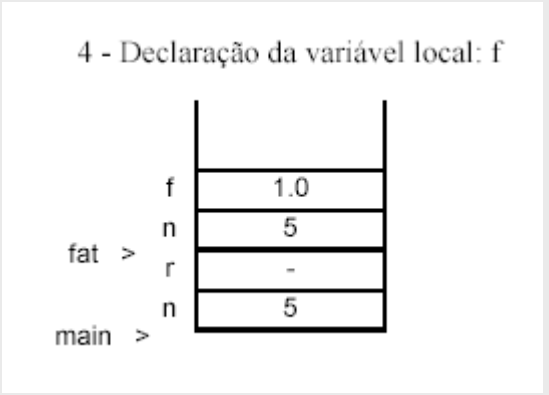
```
/* programa que lê um numero e imprime seu fatorial (versão 3) */
#include <stdio.h>
int fat (int n);
int main (void)
{  int n = 5;
   int r;
   r = fat ( n );
   printf("Fatorial de %d = %d \n", n, r);
   return 0;
}
int fat (int n)
{  int f = 1;
   while (n != 0) {
      f *= n;
      n--;
   }
   return f;
}
```

3 - Chamada da função: cópia do parâmetro



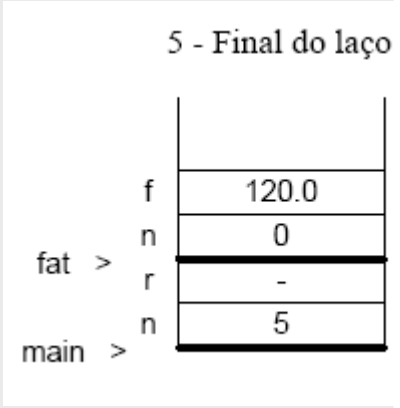
Exemplo: Declaração de `n` e `f` na `fat(int n)`

```
/* 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;
}
```



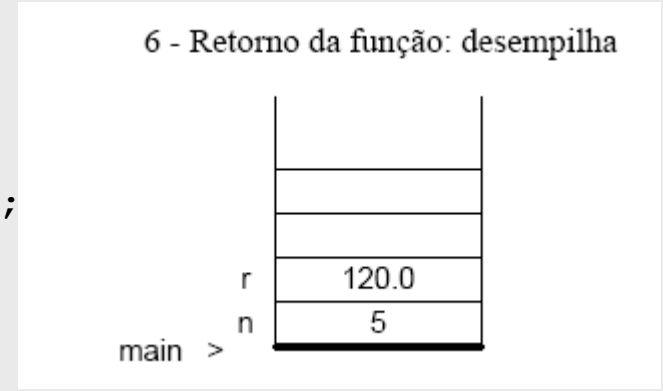
Exemplo: no final do laço

```
/* 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;
}
```



Exemplo: no retorno

```
/* 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;
}
```



Funções Recursivas

- Tipos de recursão:
  - direta:
    - uma função A chama a ela própria
  - indireta:
    - uma função A chama uma função B que, por sua vez, chama A
- Comportamento:
  - quando uma função é chamada recursivamente, cria-se um ambiente local para cada chamada
  - as variáveis locais de chamadas recursivas são independentes entre si, como se estivéssemos chamando funções diferentes

Funções Recursivas

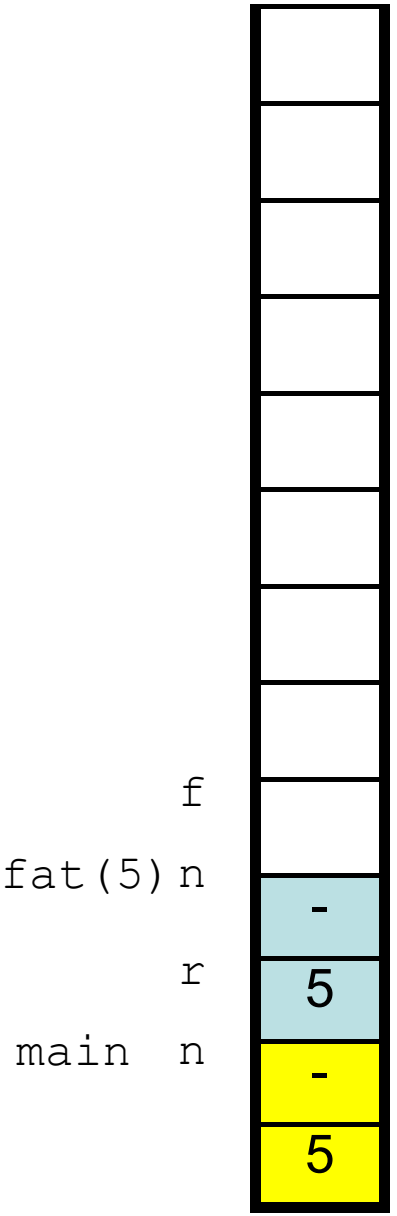
- Exemplo: definição recursiva de fatorial

$$n! = \begin{cases} 1, se\ n = 0 \\ n \times (n-1)!, se\ n > 0 \end{cases}$$

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

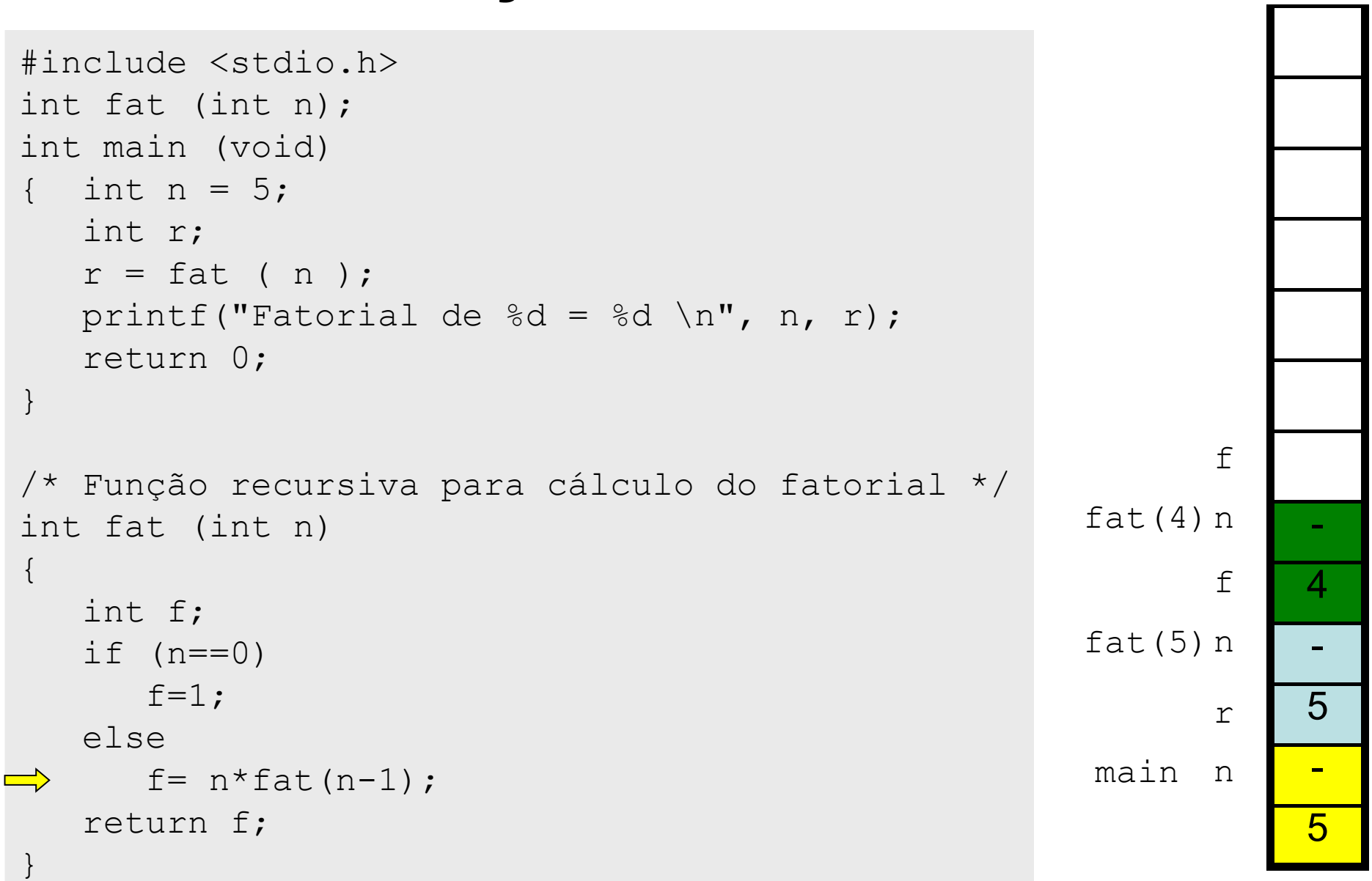
Funções Recursivas

```
#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;
}
```



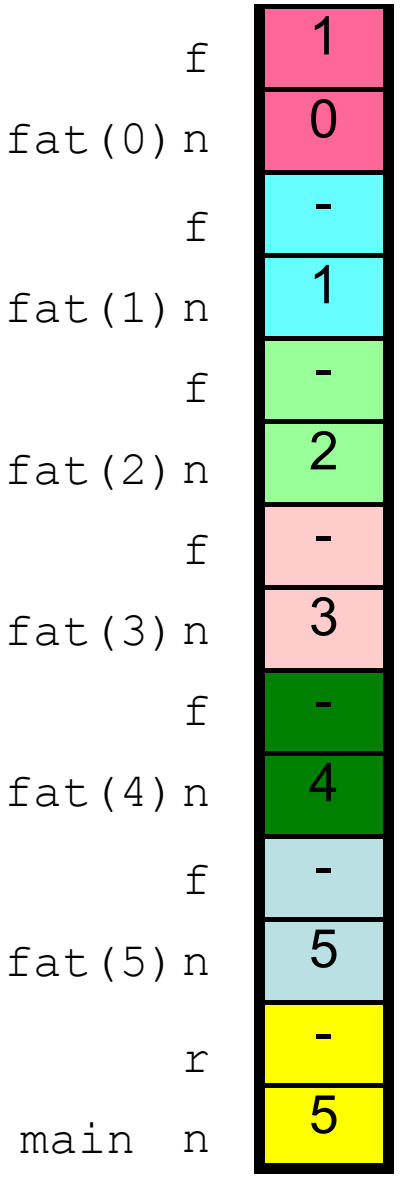
Funções Recursivas

```
#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;
}
```



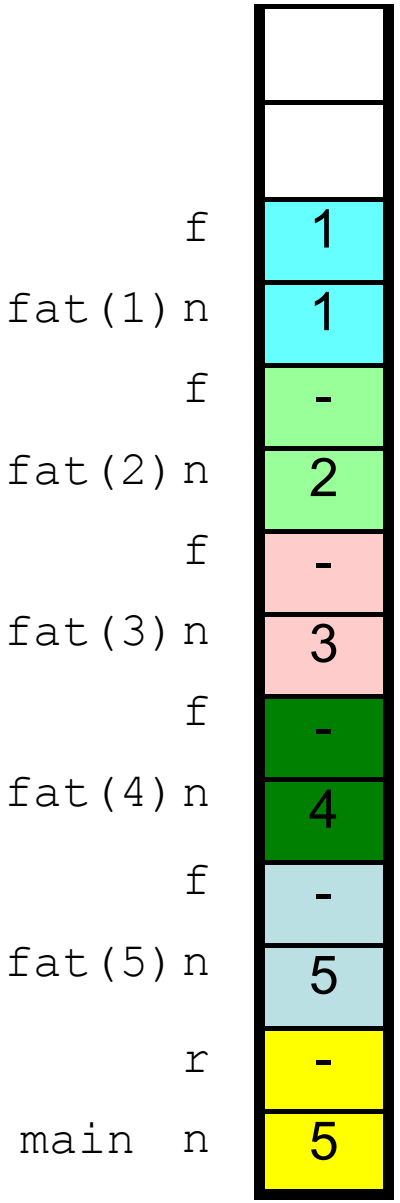
Funções Recursivas

```
#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;
}
```



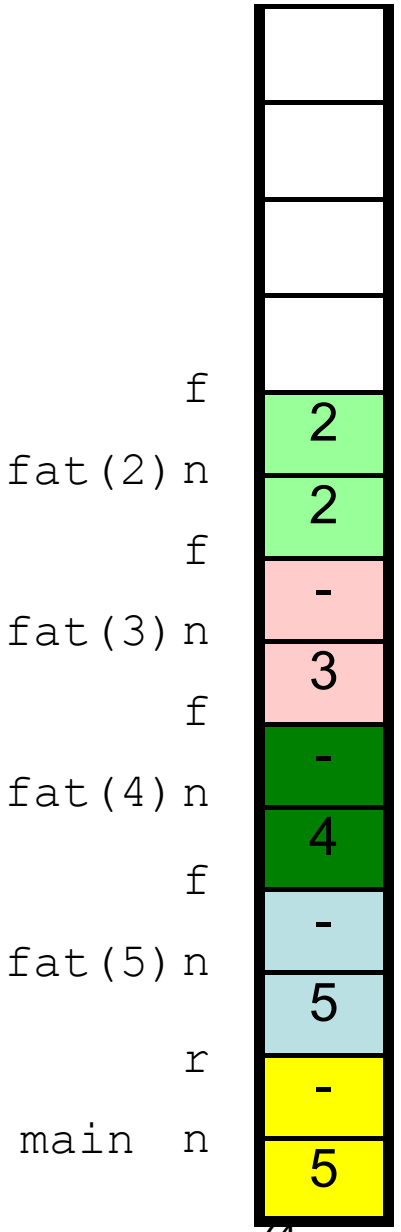
Funções Recursivas

```
#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;
}
```



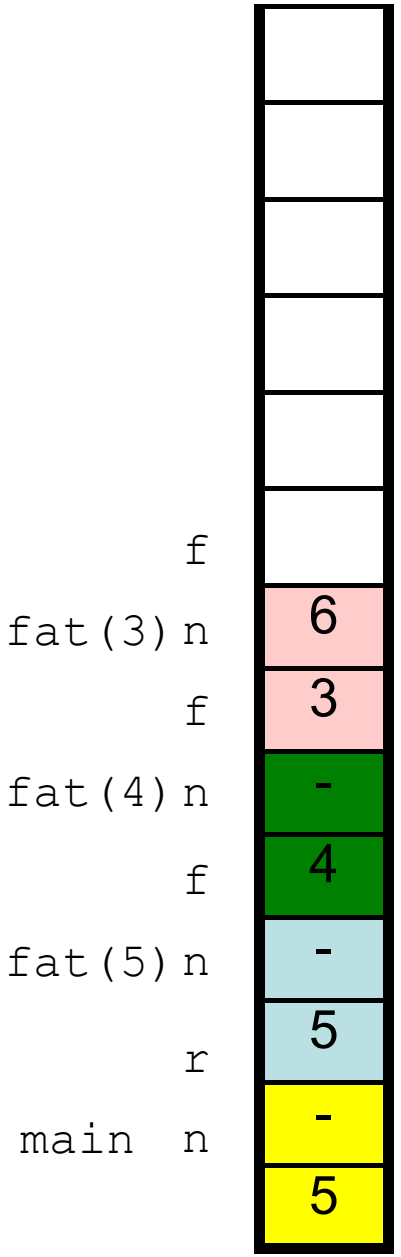
Funções Recursivas

```
#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;
}
```



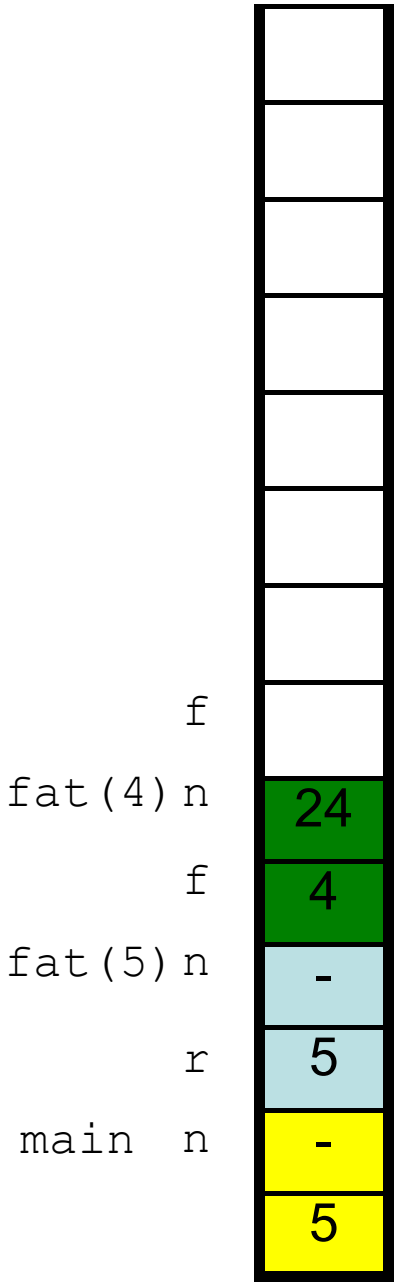
Funções Recursivas

```
#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;
}
```



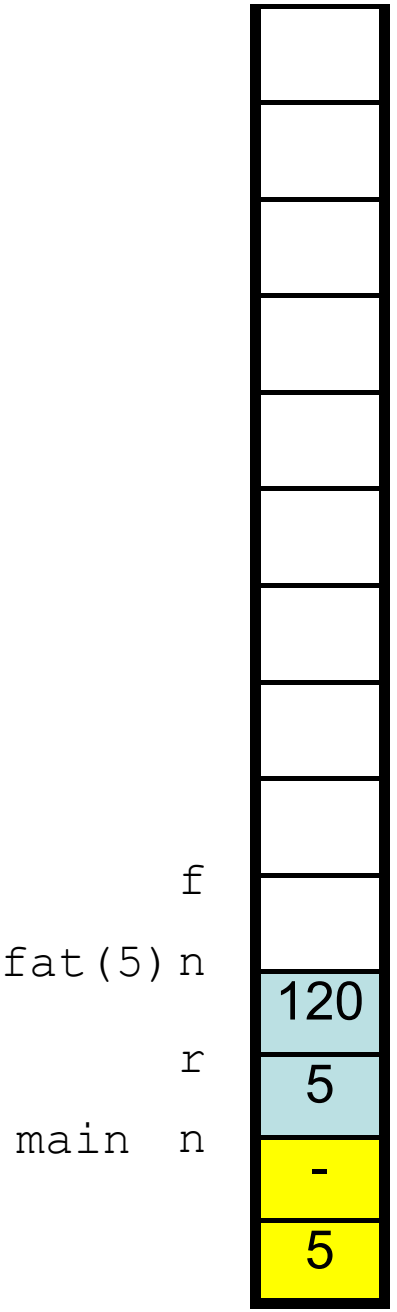
Funções Recursivas

```
#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;
}
```



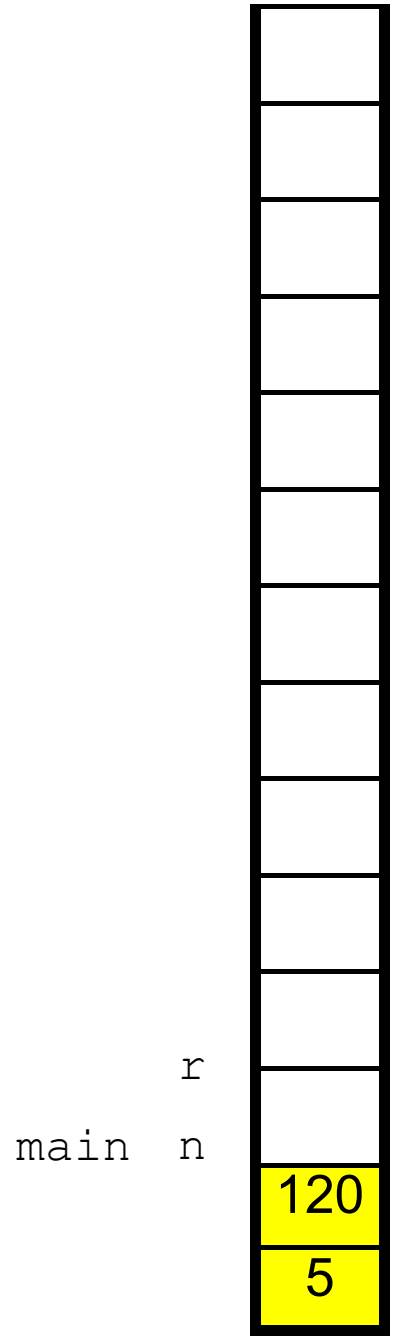
Funções Recursivas

```
#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;
}
```



Variáveis Globais

- Variável global:
  - declarada fora do corpo das funções:
    - visível por todas as funções subseqüentes
  - não é armazenada na pilha de execução:
    - não deixa de existir quando a execução de uma função termina
    - existe enquanto o programa estiver sendo executado
  - 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 em que foi declarada
  - não é armazenada na pilha de execução:
    - armazenada em uma área de memória estática
    - continua existindo antes ou 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++;
}
```

–função para imprimir números reais:

•imprime um número por vez, separando-os por espaços em branco e colocando, no máximo, cinco números por linha

Variáveis Estáticas X Globais

- variáveis estáticas e variáveis globais:
  - são inicializadas com zero,
    se não forem explicitamente inicializadas
- variáveis globais estáticas:
  - são visíveis para todas as funções subseqüentes
  - não podem ser acessadas por funções definidas
    em outros arquivos
- funções estáticas:
  - não podem ser chamadas por funções definidas
    em outros arquivos

Exercícios

- Faça um programa que recebe como entrada três graus: G1, G2 e G3 e calcula a média, se o aluno estiver aprovado, ou informa a necessidade de uma prova final, se o aluno não tiver satisfeito o seguinte critério:
  - Todas as notas maiores ou iguais a 3 E
  - Média aritmética maior ou igual a 5

- Coloque o cálculo da média em uma função separada

Exercícios

```
#include <stdio.h>

float calculaMedia(float g1, float g2, float g3);

int main(void) {
  float g1, g2, g3, media;

  printf("Digite os graus G1, G2 e G3: ");
  scanf("%f %f %f", &g1, &g2, &g3);
  media = calculaMedia(g1, g2, g3);
  if (media >= 5.0 && g1 >= 3.0 && g2 >= 3.0 && g3 >= 3.0) {
    printf("SF = APROVADO, MF = %f\n", media);
  }
  else {
    printf("ALUNO EM PROVA FINAL.\n");
  }
}

float calculaMedia(float g1, float g2, float g3) {
  float media;

  media = (g1 + g2 + g3) / 3;
  return media;
}
```

Exercícios

- Implemente uma função que retorne uma aproximação do valor de PI, de acordo com a Fórmula de Leibniz:

$$\Pi = 4 * (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + ...)$$

- Ou seja:

$$\Pi = 4 * \sum_{i=0}^{n-1} \frac{(-1)^i}{2 * i + 1}$$

- A função deve obedecer ao protótipo:
  - float pi(int n);

Exercícios

```
#include <stdio.h>

float pi(int n);

int main(void) {
  int n;
  float p;

  printf("Digite o numero de termos: ");
  scanf("%d", &n);
  if (n < 1) {
    printf("Erro! O numero de termos deve ser maior que zero.\n");
  }
  else {
    p = pi(n);
    printf("PI = %f\n", p);
  }
  return 0;
}

float pi(int n) {
  float soma;
  int i;

  soma = 1;
  for (i = 1; i < n; i++) {
    if (i % 2) {
       soma = soma - (1.0 / ((2 * i) + 1));
    }
    else {
       soma = soma + (1.0 / ((2 * i) + 1));
    }
  }
  return 4*soma;
}
```

Exercícios

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

float pi(int n);

int main(void) {
  int n;
  float p;

  printf("Digite o numero de termos: ");
  scanf("%d", &n);
  if (n < 1) {
    printf("Erro! O numero de termos deve ser maior que zero.\n");
  }
  else {
    p = pi(n);
    printf("PI = %f\n", p);
  }
  return 0;
}

float pi(int n) {
  float soma;
  int i;

  soma = 1;
  for (i = 1; i < n; i++) {
    soma = soma + (pow(-1,i) / ((2 * i) + 1));
  }
  return 4*soma;
}
```

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

- Capítulo 1 – Ciclo de Desenvolvimento
- Capítulo 2 – Expressões e E/S
- Capítulo 3 – Controle de Fluxo
- Capítulo 4 – Funções