printf:
recursos adicionais de formatação

Sinal antes de número positivo

Sinal - antes de número negativo:
sempre aparece.
Sinal + antes de números positivos:
→ deve-se usar + entre % e especificação de formato.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
  int num_posit = 10;
  int num_negat = -30;
  system("color f1");
  printf("Numero positivo: %d", num_posit);
  printf("\nNumero positivo: %+d", num_posit);
  printf("\nNumero negativo: %d", num_negat);
  printf("\nNumero negativo: %+d", num_negat);
  printf("\n");
  system("pause");
  return 0;
}
```

```
C:\backupcida\LinguagemCProgramas\FormatacaoSaida\
Numero positivo: 10
Numero positivo: +10
Numero negativo: -30
Numero negativo: -30
Pressione qualquer tecla para continuar. . .
```


Valor inteiro apresentado com zeros à esquerda

Colocar um 0 (zero) entre % e especificações outras de formato

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
  int num1 = 12,num2 = 123, num3 = 1234, num4 = 127;
  system("color f1");
  printf("Numero 1: %07d\n", num1);
  printf("Numero 2: %07d\n", num2);
  printf("Numero 3: %07d\n", num3);
  printf("Numero 4: %+07d\n", num4);
  system("pause");
  return 0;
}
```

```
C:\backupcida\Linguagem
Numero 1: 0000012
Numero 2: 0000123
Numero 3: 0001234
Numero 4: +000127
Pressione qualquer tecl
```


Texto justificado à esquerda

O padrão é ajuste pela direita, para obter ajuste pela esquerda, colocar um - (sinal de subtração) após o %

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
  int num1 = 12,num2 = 123, num3 = 1234, num4 = 127;
  system("color f1");
  printf("\nNumero 1: %-7d\n", num1);
  printf("\nNumero 2: %-7d\n", num2);
  printf("\nNumero 3: %-7d\n", num3);
  printf("\nNumero 4: %-+7d\n", num4);
  system("pause");
  return 0;
}
```

```
C:\backupcida\LinguagemC
Numero 1: 12
Numero 2: 123
Numero 3: 1234
Numero 4: +127
Pressione qualquer tecla
```

Três recursos de edição

Definição, em tempo de execução, do espaço de apresentação de um valor

Um * (asterisco) entre % e a especificação de formato, indica que o primeiro valor informado após a *string* de controle é o número de posições que devem ser usadas para apresentar o valor logo em seguida.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
  int tam = 10, valor = 12;
  system("color f1");
  printf("\n==>%*d<==  ==>%*d<==\n", 5 , 12, 10, 1);
  printf("\n==>%*d<==\n", tam , 12);
  printf("\n==>%*d<==\n", 20 , valor);
  system("pause");
  return 0;
}
```

```
C:\backupcida\LinguagemCPrograma
==>   12<==  ==>         1<==
==>        12<==
==>                  12<==
Pressione qualquer tecla para c
```


fgets:
lendo *strings* com controle de tamanho máximo

fgets

fgets(*string*, tamanho máximo da *string*, *stdin*)

Na leitura de *strings* a partir do teclado, limita o número de bytes lidos ao tamanho máximo - 1 informado.

Diferenças na leitura de strings via teclado usando gets ou fgets:

1) gets - não há controle do número de caracteres transferidos para a memória: se precisar ultrapassar a área reservada para a variável, isto será feito; além disso, o caractere correspondente ao ENTER - '\n - é substituído pelo finalizador de string - '\0' -.

2) fgets - o número de bytes transferidos para a memória está limitado pelo valor inteiro determinado em tamanho máximo indicado - 1 (o último fica reservado para inserção do '\0'), sendo que o caractere de nova linha '\n', se dentro do tamanho máximo previsto, também é incluído na *string*, em *gets* isso não ocorre.


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TAMLIN 13
int main ( )
{
 char linha1[TAMLIN], linha2[TAMLIN] = "arqteste.dat"; //contém 12 caracteres
 system("color f1");
 printf("Tamanho da linha 1 (com sizeof): %d", sizeof(linha1));
 printf("\nTamanho da linha 2 (com sizeof): %d\n", sizeof(linha2));
 printf("Informe linha 1: ");
 fgets(linha1, sizeof(linha1), stdin); // máximo estabelecido : 12 caracteres
 printf("Linha 1 = %s\n", linha1);
 printf("Comprimento da linha 1  (com strlen) eh %d", strlen(linha1));
 printf("\nLinha 2 = %s\n", linha2);
 printf("Comprimento da linha 2  (com strlen) eh %d \n", strlen(linha2));
 if (strcmp(linha1,linha2) == 0)
    printf("nomes  iguais!!!");
 else
    printf("diferentes %s %s", linha1, linha2);
 printf("\n");
 system("pause");
 return 0;
}
```

```
C:\backupcida\LinguagemCPagina2008?\AULA12FGETS
Tamanho da linha 1 (sizeof): 13
Tamanho da linha 2 (sizeof): 13
Informe linha 1: arqteste.dat
Linha 1 = arqteste.dat
Comprimento da linha 1  (strlen) eh 12
Linha 2 = arqteste.dat
Comprimento da linha 2  (strlen) eh 12
nomes  iguais!!!
Pressione qualquer tecla para continuar. . .
```

Texto lido com *fgets*, armazenado sem marca de nova linha (não tem espaço).


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TAMLIN 14
int main ( )
{
 char linha1[TAMLIN], linha2[TAMLIN] = "arqteste.dat"; //contém 12 caracteres
 system("color f1");
 printf("Tamanho da linha 1 (com sizeof): %d", sizeof(linha1));
 printf("\nTamanho da linha 2 (com sizeof): %d\n", sizeof(linha2));
 printf("Informe linha 1: ");
 fgets(linha1, sizeof(linha1), stdin); // máximo estabelecido: 13 caracteres
 printf("Linha 1 = %s", linha1);
 printf("Comprimento da linha 1  (com strlen) eh %d", strlen(linha1));
 printf("\nLinha 2 = %s\n", linha2);
 printf("Comprimento da linha 2  (com strlen) eh %d \n", strlen(linha2));
 if (strcmp(linha1,linha2) == 0)
    printf("nomes  iguais!!!");
 else
    printf("diferentes %s %s", linha1, linha2);
 printf("\n");
 system("pause");
 return 0;
}
```

```
C:\backupcida\LinguagemCPagina2008?\AULA12FGETS
Tamanho da linha 1 (sizeof): 14
Tamanho da linha 2 (sizeof): 14
Informe linha 1: arqteste.dat
Linha 1 = arqteste.dat
Comprimento da linha 1  (strlen) eh 13
Linha 2 = arqteste.dat
Comprimento da linha 2  (strlen) eh 12
diferentes arqteste.dat
arqteste.dat
Pressione qualquer tecla para continuar. . .
```

Texto lido com *fgets*, armazenado com marca de nova linha (tem espaço).

Eliminação

da marca de nova linha, se houver,

de uma *string* lida com *fgets*


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TAMLIN 14
int main ( )
{
 char linha1[TAMLIN], linha2[TAMLIN] = "arq";
 system("color f1");
 printf("Tamanho da linha 1 (com sizeof): %d", sizeof(linha1));
 printf("\nTamanho da linha 2 (com sizeof): %d\n", sizeof(linha2));
 printf("Informe linha 1: ");
 fgets(linha1, sizeof(linha1), stdin); // caracteres lidos
 printf("Linha 1 = %s", linha1);
 printf("Comprimento da linha 1  (com strlen) eh %d", strlen(linha1));
 if (linha1[strlen(linha1) - 1] == '\n') // se couber, o \n é incluído antes do \0
    linha1[strlen(linha1) - 1] = '\0'; // substitui \n por \0 (ficam 2 \0 seguidos)
 printf("\nComprimento da linha 1  (com strlen) eh %d", strlen(linha1));
 printf("\nLinha 2 = %s\n", linha2);
 printf("Comprimento da linha 2  (com strlen) eh %d \n", strlen(linha2));
 if (strcmp(linha1,linha2) == 0)
    printf("nomes  iguais!!!");
 else
    printf("diferentes %s %s", linha1, linha2);
 printf("\n");
 system("pause");
 return 0;
}
```

```
C:\backupcida\LinguagemCPagina2008?\AULA12FGETSCida
Tamanho da linha 1 (com sizeof): 14
Tamanho da linha 2 (com sizeof): 14
Informe linha 1: arq
Linha 1 = arq
Comprimento da linha 1  (com strlen) eh 4
Comprimento da linha 1  (com strlen) e sem \n eh 3
Linha 2 = arq
Comprimento da linha 2  (com strlen) eh 3
nomes  iguais!!!
Pressione qualquer tecla para continuar. . .
```

Eliminando o \n de uma *string* lida com *fgets*

Revisando:

- **strlen** devolve o número de caracteres de uma *string*, sem contar o **\0** (caractere finalizador de *string*).
- **sizeof** retorna o tamanho de um tipo ou da área reservada para uma variável, em bytes.

Os índices para acessar os elementos de um vetor iniciam em 0, o que significa que a posição contendo **\0** é a strlen(var_string) **- 1**.

Por isso:

linha1[strlen(linha1) - 1] = '\0'; **// insere finalizador**

O número máximo válido de caracteres obtidos no preenchimento de uma string é sizeof(var_string) .

Por isso:

fgets(linha1, sizeof(linha1), stdin); /* lê no máximo TAMANHO - 1 caracteres, inserindo o caractere enter(\n) antes do finalizador *\

Subprogramas

- conceito de subprogramação
- funções void (sem retorno)
- funções que retornam valores

Supondo o problema:

Algoritmo em passos gerais:

1. início
2. preencher vetor a(MAX) com valores do índice
3. imprimir vetor a
4. somar 2 em cada elemento de a
5. imprimir vetor a
6. zerar conteúdos pares
7. imprimir vetor a
8. fim


```
...
int main()
{
  int a[MAX], k;
  //preencher com valores dos índices
  for (k=0; k<MAX; k++)
          a [k] = k;
    //imprimir a
  for (k=0; k<MAX; k++)
          printf("%d\n",a[k]);
 //somar 2 em cada elemento
  for (k=0; k<MAX; k++)
          a [k] = a[k] + 2;
  //imprimir a
  for (k=0; k<MAX; k++)
          printf("%d\n",a[k]);
  //zerar os pares
  for (k=0; k<MAX; k++)
          if (a[k] % 2 == 0)
            a[k] = 0;
  //imprimir a
  for (k=0; k<MAX; k++)
          printf("%d\n",a[k]);
...
system("pause");
}
```


Subprograma

Objetivos principais:

- ✓ **evitar repetição de seqüência de comandos.**
- ✓ **dividir e estruturar um programa em partes fechadas e logicamente coerentes.**

> **"A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas"**
>
> **Dijkstra, 1972**

Programação Estruturada

- desenvolvimento de algoritmos por fases ou refinamentos sucessivos
- uso de um número muito limitado de estruturas de controle
- decomposição do algoritmo total em módulos

"dividir para conquistar"

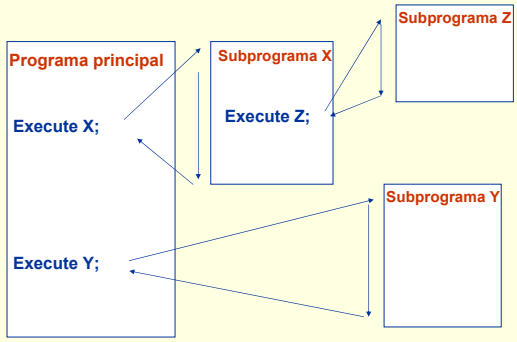

```
int main()
{
  int a[MAX], k;
  //preencher com valores dos índices
  for (k=0; k<MAX; k++)
          a [k] = k;
    //imprimir a
  for (k=0; k<MAX; k++)
          printf("%d\n",a[k]);
 //somar 2 em cada elemento
  for (k=0; k<MAX; k++)
          a [k] = a[k] + 2;
  //imprimir a
  for (k=0; k<MAX; k++)
          printf("%d\n",a[k]);
  //zerar os pares
  for (k=0; k<MAX; k++)
          if (a[k] % 2 == 0)
            a[k] = 0;
  //imprimir a
  for (k=0; k<MAX; k++)
          printf("%d\n",a[k]);
...
system("pause");
}
```

```
int main()
{
  int a[MAX], k;
  //preencher com valores dos índices
  for (k=0; k<MAX; k++)
          a [k] = k;
//imprimir a
execute imprimirvetor

//somar 2 em cada elemento
  for (k=0; k<MAX; k++)
          a [k] = a[k] + 2;

  //imprimir a
  execute imprimirvetor

  //zerar os pares
  for (k=0; k<MAX; k++)
          if (a[k] % 2 == 0)
            a[k] = 0;
//imprimir a
 execute imprimirvetor
...
system("pause");
}
```

Subprogramas

- programas independentes, mas executados somente quando chamados por outro programa.
- após sua execução, o fluxo do programa retorna ao ponto imediatamente após o da chamada do subprograma.
- devem executar UMA tarefa específica, muito bem identificada (programação estruturada).
- podem ser testados separadamente.
- permitem criar bibliotecas de subprogramas.
- tipos: funções sem retorno, funções com retorno.

Vários Níveis de Chamada



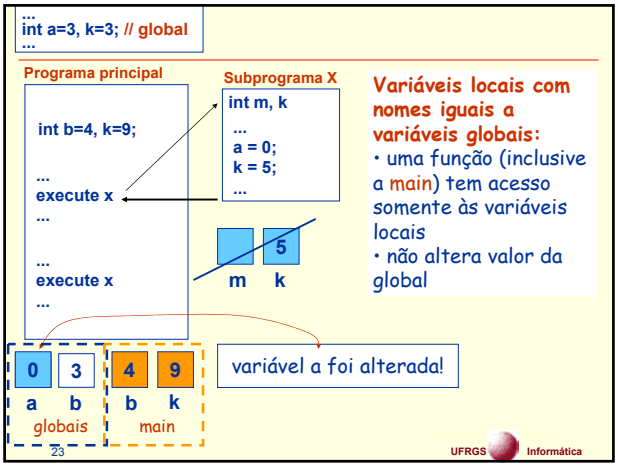
Variáveis Globais X Variáveis Locais

- Globais:
  - declaradas fora das funções (inclusive da *main*);
  - reconhecidas pelo programa inteiro;
  - podem ser usadas em qualquer trecho de código;
  - existem durante toda a execução do programa.
  - uso limitado (programação estruturada).
- Locais:
  - declaradas dentro de uma função;
  - só podem ser referenciadas por comandos que estão dentro do bloco (função) no qual elas foram declaradas;
  - existem apenas enquanto o bloco de código em que foram declaradas está sendo executado.

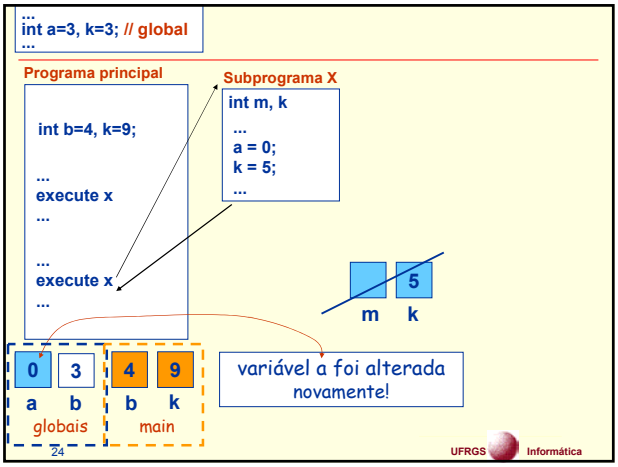
Variáveis Globais X Variáveis Locais

```
/* comentário inicial, descrevendo objetivos do
programa */
#include <stdlib.h> // para usar system
#include <stdio.h> // para entrada e saída de dados
// outras bibliotecas necessárias
// outras declarações, incluindo variáveis globais
// funções ou protótipos de funções
int main( ) // função principal e obrigatório
{
   //declarações de variáveis locais e comandos//

   system ("pause");   /* ou system("pause >> null");
                          - para eliminar mensagem  */
   return 0; // para encerramento normal do programa
}
// funções com protótipo
```


Variáveis Globais X Variáveis Locais



Variáveis locais com nomes iguais a variáveis globais:
- uma função (inclusive a main) tem acesso somente às variáveis locais
- não altera valor da global



função soma2
/ faz a soma de dois valores a e b, e retorna a soma na variável soma */*
soma = a + b
fim função soma2

Como usar *soma2* para:
somar a com b
somar a com c ?
somar b com c ?

Parâmetros Formais e Reais

subprograma soma2 (v1, v2)
{ parâmetros inteiros v1 e v2}
soma = v1 + v2
fim subprograma soma2

Parâmetros formais
(valores a serem fornecidos na execução)

variável global soma // para a soma
algoritmo somar
variáveis: a, b, c // para valores lidos - inteiros
início
ler a, b, c
executar soma2 (a, b) // executa soma2 com os valores de a e b
escrever soma
executar soma2 (b, c) // executa soma2 com os valores de b e c
escrever soma
executar soma2 (a, c) // executa soma2 com os valores de a e c
escrever soma
fim.

Parâmetros reais
ou argumentos
(utilizados na execução)

Parâmetros Formais e Reais

subprograma soma2 (int x, int y)
// parâmetros formais x e y são inteiros
soma = x + y
fim subprograma soma2

Parâmetros formais (valores a serem fornecidos na execução)

programa somavarios
variáveis a,b,c – inteiros
soma – inteiro
v(10) – inteiros
início
ler a, b, c
soma2 (a, b)
escrever soma
soma2 (a, c)
escrever soma
para i de 0 a 9 faça
ler v[i]
para i de 0 a 8 faça
execute soma2 (v[i], v[i+1])
escrever soma
...

Parâmetros reais
ou argumentos
(utilizados na execução)

parâmetros reais (ou argumentos) devem concordar em número e tipo com os formais, na ordem definida.

Subprogramas

- conceito de subprogramação
- funções void (sem retorno)
- funções que retornam valores

Programa principal

```
int main() // esta função não é void
{
 int a[MAX], k;
 //preencher com valores dos índices
 for (k=0; k<MAX; k++)
        a [k] = k;
 //imprimir a
 execute imprimirvetor

 //somar 2 em cada elemento
 for (k=0; k<MAX; k++)
        a [k] = a[k] + 2;

 //imprimir a
 execute imprimirvetor

 //zerar os pares
 for (k=0; k<MAX; k++)
        if (a[k] % 2 == 0)
          a[k] = 0;
 //imprimir a
 execute imprimirvetor
...
system("pause");
}
```

Subprograma

```
void imprimirvetor(void)
{
for (k=0; k<MAX; k++)
        printf("%d\n",a[k]);
}
```

void : indica função sem retorno;
void: indica função sem parâmetros
→omissão de tipo de função ou de parâmetro indica substituição por tipo ou argumento inteiro.

Funções sem retorno (procedimento)

// void indica sem retorno de valor associado à execução da função:
void identificador **(void** ou lista de parâmetros formais**)**
{
[declaração de variáveis locais**;]**
comandos separados por ';'
}

opcionais

lista de parâmetros formais:
(tipo identificador1, tipo identificador2, ...)
Sem parâmetros:
(void) ou ( )

```
void teste ( int a, int b,int c )
// exemplo da sintaxe de procedimento
{
   int x;
   scanf("%d", &x);
   c = a + b + x;
   printf("\n Resultado = %d', c);
}
```

Declaração de funções void, sem uso de protótipo

- Devem ser declaradas antes de serem usados.

```
//programa exemplo
// ****************************************
void um (int x)
{ escreve o valor do parâmetro }
{
   printf("%d", x);
}
// ****************************************
int main()
{
   int a,b;
   scanf("%d%d",&a,&b)
   um (a);
   um (a + b);
}
```

Declaração da função, antes de ser usada.

Declaração de funções void, com uso de protótipo

- Se não estiverem declaradas antes do programa principal, incluir pré-declaração através do recurso do protótipo.

Protótipo da função

Chamada da função

Declaração da função

```
#include <stdio.h>
#include <stdlib.h>
void escreve_linha(void); // protótipo: com ;

int main()
{
   escreve_linha();
   system("pause");
}

void escreve_linha(void) // declaração: sem ;
{
    int c;
    for (c=1; c<=80; c++)
      printf("*");
}
```


Exercício 1:

Faça um procedimento que receba 2 parâmetros: um caractere **c** e um inteiro **n**. O procedimento deverá imprimir na tela **n** vezes o caractere **c**.

Solução 1 – exercício 1

```
#include <stdio.h>
#include <stdlib.h>
void imprime(char, int); // protótipo: apenas com tipo dos parâmetros
int main()
 {
   int num=10;
   char letra='a';
   imprime(letra,num);
   system("pause");
   return 0;
 }
void imprime(char c, int n) // declaração da função void
{
    int cont; // variável local de controle do laço
    for (cont = 0; cont < n; cont++)
        printf("%c",c);
}
```

Chamada da função

Solução 2 – exercício 1

```
#include <stdio.h>
#include <stdlib.h>
void imprime(char, int); // protótipo apenas com tipo dos parâmetros
int main()
 {
   int num=10;
   char letra='a';
   imprime(letra,num);
   system("pause");
   return 0;
 }
void imprime(char c, int n) // declaração da função void
{
    for (; n> 0; n--)   // usar n aqui não vai alterar argumento num
        printf("%c",c);
}
```

Chamada da função

Variáveis Globais

- São declaradas fora das funções, após declaração de bibliotecas (antes da função main)
- São reconhecidas pelo programa inteiro
- Podem ser usadas em qualquer trecho de código
- Existem durante toda a execução do programa

Variáveis Locais

- São declaradas dentro de uma função.
- Só podem ser referenciadas por comandos que estão dentro do bloco (função) no qual elas foram declaradas – ou seja, não são reconhecidas fora da função onde foram declaradas (programa principal ou outra funções).
- Existem apenas enquanto o bloco de código em que foram declaradas está sendo executado.

Variáveis Locais e Globais – exemplo 1

```
#include <stdio.h>
#include <stdlib.h>
int cont=5; // variável global

void funcao1()
{
   int i;
   for (i=1;i < cont; i++)
          printf(".");
   printf("\n cont funcao 1  = %
}

int main()
{
    int  x;
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1();
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

```
C:\Cora\Disciplinas\INF
 cont main = 5
....
 cont funcao 1  = 5
 cont main apos funcao1 = 5
Pressione qualquer tecla para conti
```


Variáveis Locais e Globais – exemplo 2

```
#include <stdio.h>
#include <stdlib.h>
int cont=5; // variável global
void funcao1()
{
   int i;
   for (i=1;i < cont; i++) // reconhece a variável global
          printf(".");
   printf("\n cont funcao 1  = %d", cont);
}

int main()
{
    int  cont=10, x; // cont local, do programa principal
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1();
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

```
C:\Cora\Disciplinas\INFO12...
 cont main = 10
....
 cont funcao 1  = 5
 cont main apos funcao1 = 10
Pressione qualquer tecla para continuar. .
```


Variáveis Locais e Globais – exemplo 3

```
#include <stdio.h>
#include <stdlib.h>
int cont=5; // variável global
void funcao1()
{
   int cont=15, i; // cont local da funcao1
   for (i=1;i < cont; i++)
          printf(".");
   printf("\n cont funcao 1  = %d", cont);
}

int main()
{
    int  cont=10, x; // cont local do programa principal
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1();
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

```
C:\Cora\Disciplinas\INFO12...
 cont main = 10
.............
 cont funcao 1  = 15
 cont main apos funcao1 = 10
Pressione qualquer tecla para continuar. .
```


Variáveis Locais e Globais – exemplo 4

```
#include <stdio.h>
#include <stdlib.h>
// cont global foi excluída!!!
void funcao1( )
{
   int i; // declaração cont local também foi excluída
   for (i=1;i < cont; i++)
          printf(".");
   printf("\n cont funcao 1  = %d", cont);
}

int main()
{
    int  cont=10, x; // cont local do programa principal - não é global
    system("color 71");
    printf("\n cont main = %d \n", cont);
    funcao1();
    printf("\n cont main apos funcao1 = %d \n", cont);
    system("pause");
    return(0);
}
```

O que aconteceria?

Programa não compila!

Importante

- Em subprogramas, utilizar somente variáveis locais.
- Troca de dados entre módulos feita somente através de parâmetros!
- Identificar claramente os parâmetros de entrada (que fornecem valores para serem processados) e os de saída (que retornam o que foi calculado ao programa principal)
- NUNCA UTILIZAR VARIÁVEIS GLOBAIS!!!! (exceções a serem vistas mais adiante)