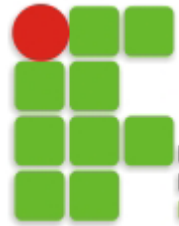


Análise e Desenvolvimento de Sistemas

Instituto Federal da Bahia

Análise e Desenvolvimento de Sistemas

INF029 – Laboratório de Programação

Aula 02: Modularização

Prof. Dr. Renato L. Novais
renato@ifba.edu.br

Agenda

- Modularização
- Funções em C

Modelo inicial de desenvolvimento

- Todo o código em um único arquivo e em um único bloco (e.g. *int main()*)

- Alguns problemas
  - Códigos grandes
  - Difícil entendimento
  - Repetição de partes do código
  - Difícil Manutenção
  - Evita trabalho em equipe
  - Etc.

- Prática ruim de programação que conduz ao insucesso do seu software
  - A crise do software (Década de 1970)

Softwares complexos e grandes são difíceis de construir em um único arquivo.

A manutenção também será prejudicada

Um dos objetivos da modularização

Facilitar a solução de problemas complexos.

"A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas"

Dijkstra, 1972

Divida para conquistar

(Divide and Conquer)

Divisão de um problema original em subproblemas (módulos) mais fáceis de resolver e transformáveis em trechos mais simples, com poucos comandos (subprogramas).

Modularização

- É o grau no qual os componentes dos sistemas podem ser separados e recombinados
- Tem como objetivo fazer a decomposição do código fonte em módulos
- Dividir a complexidade do problema em partes
- Organizar o processo de programação

Módulo

- É um grupo de comandos (trecho de algoritmo) com uma funcionalidade bem definida e o mais independente possível em relação ao resto do programa

- Separação de interesses (Separation of concerns)

Na grande maioria dos sistemas de software, as **modificações executadas são pontuais**, o que não permite que se compreenda a completa natureza do sistema (Parnas, 1994)

Modularização

Facilita a compreensão

Melhora a Reusabilidade

Modular programming

- No passado, programação modular era vista como um meio de construir programas em pequenos pedaços: "sub-rotinas"

- Mas modularidade não traz benefícios a menos que os módulos sejam
  - Autônomos,
  - Coerentes e
  - Robustos

Subprogramas

- Trechos de código independentes, com estrutura semelhante àquela de programas, mas executados somente quando chamados por outro(s) trecho(s) de código.

- Devem executar UMA tarefa específica, muito bem identificada (conforme a programação estruturada).

- Ao ser ativado um subprograma, o fluxo de execução desloca-se do fluxo principal para o subprograma. Concluída a execução do subprograma, o fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada do subprograma.

Vantagens do uso de subprogramas

- Maior controle sobre a complexidade
- Estrutura lógica mais clara
- Maior facilidade de depuração e teste, já que subprogramas podem ser testados separadamente.
- Possibilidade de reutilização de código

Exemplos de módulos em programação

- Arquivo
- Função
- Procedimento
- Pacote
- Classe
- Método
- ...

Modularização em C

- Funções

Funções

- Segmentos de programa que executam uma determinada tarefa específica.
- Funções (também chamadas de rotinas, ou sub-programas) são a essência da programação estruturada.
- Ex: sqrt(), strlen(), etc.
- O programador também pode escrever suas próprias funções, chamadas de funções de usuário, que tem uma estrutura muito semelhante a um programa.

Forma geral da declaração de uma função

```
tipo_da_funcao    nome_da_função (lista_de_parâmetros)
{
    //declarações locais
   //comando
}
```

- tipo_da_funcao: o tipo de valor retornado pela função. Se não especificado, por falta a função é considerada como retornando um inteiro.
- nome_da_função: nome da função conforme as regras do C
- lista_de_parâmetros: tipo de cada parâmetro seguido de seu identificador, com vírgulas entre cada parâmetro. Mesmo se nenhum parâmetro for utilizado, os parênteses são obrigatórios.
  - Os parâmetros da declaração da função são chamados de parâmetros formais.

Exemplos de cabeçalhos de funções

Ex.:

```
soma_valores (int valor1, int valor2) // por falta é inteira

void imprime_linhas(int num_lin)

void  apresenta_menu (  )

float conv_dolar_para_reais(float dolar);
```

Funções void

Void é um termo que indica ausência.

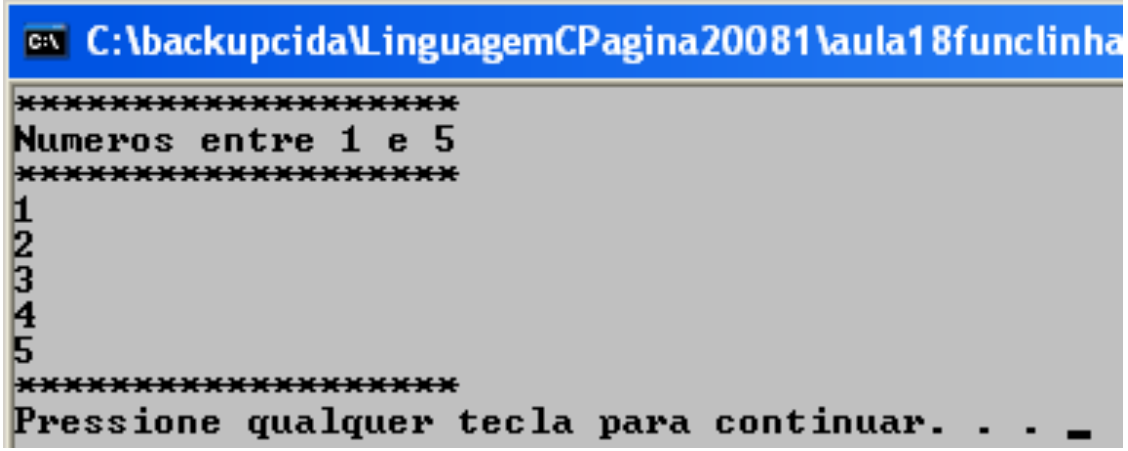
Em linguagem C é um tipo de dados.

Programa escreveint versão 1:

com linha de asteriscos produzida por trecho que se repete no código.

```
//Escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
main( )
{
   int i;
   system("color70")
   //apresentacao do cabecalho
   for (i=1;i<20;i++)
        printf("*");
   printf("\n");
   printf("Numeros entre 1 e 5\n");
   for (i=1;i<20;i++)
        printf("*");
   printf("\n");
   //escrita dos numeros
   for (i=1;i<=5;i++)
        printf("%d\n",i);
   for (i=1;i<20;i++)
        printf("*");
   printf("\n");
   system("pause");
}
```

Execução



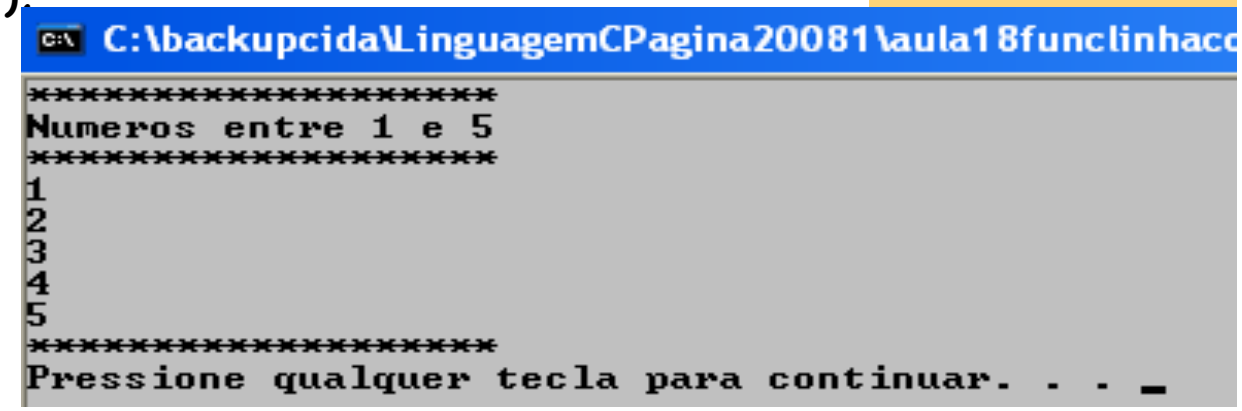
- A repetição de trechos de código idênticos em um programa pode ser um procedimento fácil e rápido, mas facilmente tende a produzir erros.
- A Manutenção com trechos repetidos tende a ser mais trabalhosa e sujeita a erros.
- Alterações de trechos iguais que se repetem não são realizadas em todas as ocorrências do trecho ou são realizadas de forma incompleta em alguma ocorrência, com resultados bastante danosos.
- A solução para esta questão são os subprogramas.
- A seguir uma versão do programa **escreveint** onde as linhas de asterisco são produzidas pela função apresente_linha.

Programa escreveint versão 2:

com uma função para apresentar linhas de asteriscos.

```
//escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
void apresente_linha(void);
main( )
{
    int i;
    system("color 70");
    //apresentacao do cabecalho
    apresente_linha( );
    printf("Numeros entre 1 e 5\n");
    apresente_linha( );
    // Escrita dos numeros
    for (i=1;i<=5;i++)
         printf("%d\n",i);
    apresente_linha( );
    system("pause");
}
void apresente_linha (void)
 {
  int i;
  for (i=1;i<20;i++)
       printf("*");
  printf("\n");
 }
```

Execução



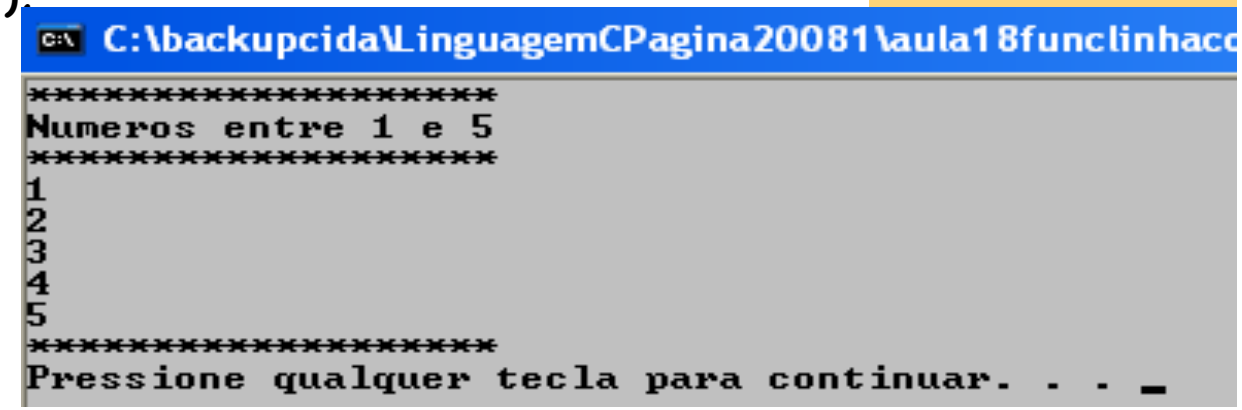
Chamadas à função apresente_linha substituem trechos repetidos

```
//escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
void apresente_linha(void);
main( )
{
    int i;
    system("color 70");
    //apresentacao do cabecalho
    apresente_linha( );
    printf("\Numeros entre 1 e 5\n");
    apresente_ linha( );
    // Escrita dos numeros
    for (i=1;i<=5;i++)
        printf("%d\n",i);
    apresente_linha( );
    system("pause");
}
```

Protótipo da função apresente_linha

Chamadas da função apresente_linha

Execução




```
void apresente_ linha (void)
 {
  int i;
  for (i=1;i<20;i++)
       printf("*");
  printf("\n");
 }
```

Declaração da função apresente_linha:
escreve uma linha de asteriscos.

Cabeçalho da função apresente_linha

```
void apresente_linha (void)
```

Indica que a função não retorna valor no seu nome.

Indica que a função não tem parâmetros.

A função apresente_linha realiza sua tarefa sem receber nenhum valor do mundo externo à função, via parâmetros, e sem retornar nenhum valor no seu nome.

Seu tipo é void e seus parâmetros são void.

Exercícios

- Faça um programa que tenha as seguintes opções de menu (cada um deve ser uma função)
  - A) Imprimir hora do sistema
  - B) Imprimir data do sistema
  - C) Somar dois números. Os números devem ser solicitados ao usuário dentro da função
  - D) Imprima os últimos dois valores digitados pelo usuário e o resultado da soma

Execução de uma função

- Ao chamar uma função, a execução é desviada para o trecho de código da função.
- A função é ativada e os itens locais à função são criados (os parâmetros por valor e os itens declarados internamente à função, como variáveis e constantes).
- A função é executada até que seu término seja atingido.
- Concluída a execução da função, todos os elementos locais à função que foram criados no momento de sua ativação são destruídos e a execução retorna ao fluxo principal, ao ponto imediatamente seguinte àquele onde ocorreu a chamada da função.

Variáveis locais

Os parâmetros que aparecem no cabeçalho das funções e as variáveis e constantes declaradas internamente a funções são locais à função.
Na função apresente_linha, o i é uma variável local a essa função.

```
void apresente_ linha (void)
 {
    int i;
    for (i=1;i<20;i++)
         printf("*");
    printf("\n");
 }
```

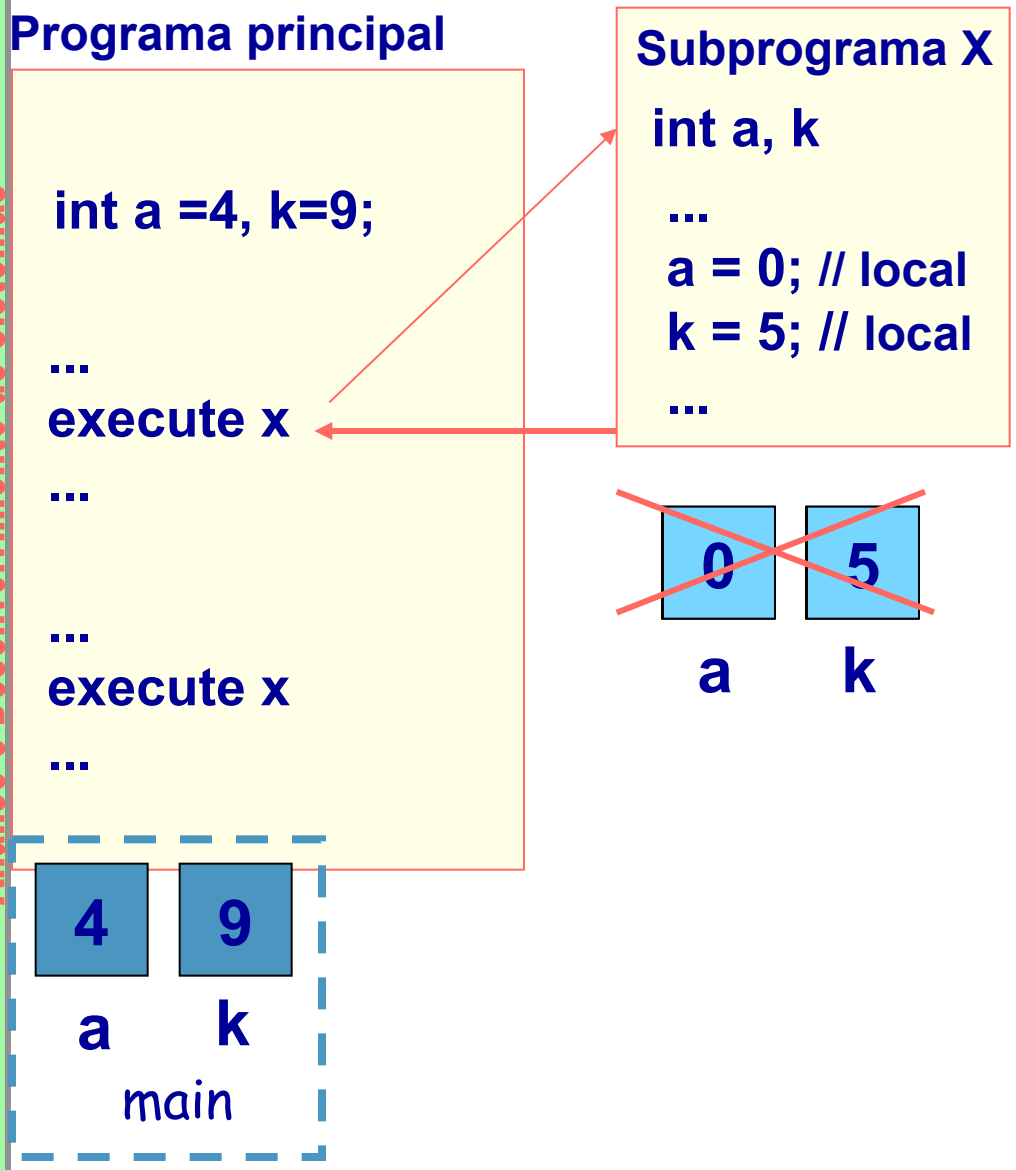
Importante:

Recomenda-se fazer todas as declarações de uma função no seu início.

As variáveis e constantes declaradas em uma função são ditas locais à função porque:

- só podem ser referenciadas por comandos que estão dentro da função em que foram declaradas;
- existem apenas enquanto a função em que foram declaradas está sendo executada. São criadas quando a função é ativada e são destruídas quando a função encerra.

Criação e destruição de variáveis locais a uma função:



Variáveis locais :

- uma função (inclusive a main) tem acesso somente às variáveis locais
- não altera valor de variáveis de outras funções

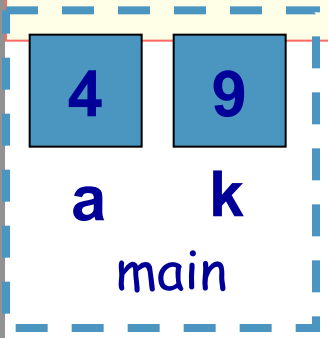
Criação e destruição de variáveis locais a uma função:

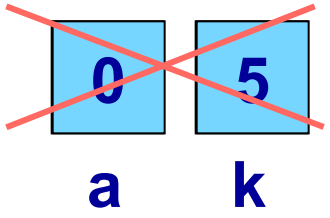
Programa principal

```
int a =4, k=9;

...
execute x
...

...
execute x
...
```

Subprograma X

```
int a, k

...
a = 0; // local
k = 5; // local
...
```



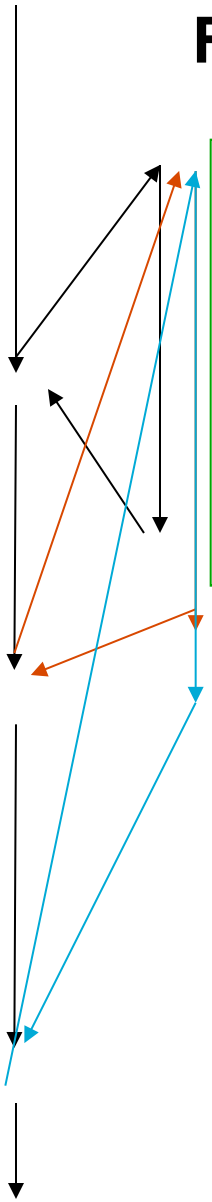


ATENÇÃO:
Uma função (inclusive a *main*) tem acesso somente às suas variáveis locais.

Função *main*

```
main( )
{
   int i;

   //apresentacao do cabecalho


   apresente_linha(  );

   printf("Numeros entre 1 e 5\n");


   apresente_linha(  );


   //escrita dos numeros
   for (i=1;i<=5;i++)
        printf("%d\n",i);

   apresente_linha (  );
   system("pause");

}
```

Execução de *main* com chamadas à função apresente_linha



Função apresente_linha

```
void apresente_linha (void)
 {
   int i;
   for (i=1;i<20;i++)
         printf("*");
   printf("\n");
 }
```

IMPORTANTE:

O i é local a apresente_linha. A cada nova execução de apresente_linha um novo i é criado e, ao final, destruído.

Funções de tipo void

- São ativadas como se fossem comandos.
- Não ocorrem dentro de expressões.
- Correspondem aos procedimentos de outras linguagens (Pascal, etc.).

Funções com tipo não void e com parâmetros

sqrt: função pré-definida

- A seguir um programa que extrai a raiz quadrada de um número indeterminado de valores informados.

- Para extrair a raiz quadrada dos valores é usada a função pré-definida sqrt, da biblioteca math.h.

sqrt: função pré-definida (cont.)

```
//extrai a raiz quadrada de valores informados
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main ( )
{
   int seguir;
   double valor;
   do
     {
       printf("\nValor para extrair raiz: ");
       scanf("%lf", &valor);
       printf ("\nRaiz quadrada de %6.2lf = %6.2lf\n",valor, sqrt(valor));
       printf("\nMais um valor, digite 1, para parar, digite 0: ");
       scanf("%d", &seguir);
     }
   while (seguir);
   system("pause");
}
```

sqrt: função pré-definida (cont.)

- A função sqrt é do tipo double, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor double.

- Para executar essa função é necessário fornecer um parâmetro, o valor para o qual se deseja que a raiz quadrada seja calculada.

- No exemplo, está armazenado na variável valor.

Exercícios

- Faça um programa que tenha as seguintes opções de menu (cada um deve ser uma função)
  - A) Imprimir hora do sistema
  - B) Imprimir data do sistema
  - C) Somar dois números. Os números devem ser solicitados ao usuário dentro da função
  - D) Imprima os últimos dois valores digitados pelo usuário e o resultado da soma
  - E) Fazer a soma de dois números e retornar o resultado, imprimir o resultado na função principal
  - F) Fazer as outras operações básicas matemáticas: subtração, multiplicação, divisão
  - G) Fazer uma função multiplicacao_por_soma, que faça a multiplicação através da soma. A função deve chamar a função de soma feito na letra E.
    - Ex: 5 x 7 = ((((5 + 5) + 5) + 5) + 5) + 5

calc_produto: função definida pelo usuário

- A seguir um programa que calcula o produto de um número indeterminado de pares de valores informados.
- Para calcular os produtos é usada a função definida pelo usuário calc_produto.

produto: função definida pelo usuário (cont.)

```
//calcula produtos de pares de valores informados
#include <stdio.h>
#include <stdlib.h>
int calc_produto(int, int);
main (  )
{
    int seguir;
    int oper1, oper2, produto;
    do
      {
        printf("\nOperando 1: ");
        scanf("%d", &oper1);
        printf("\nOperando 2: ");
        scanf("%d", &oper2);
        printf ("\nProduto = %d\n", calc_produto(oper1, oper2));
        printf("\nPara continuar, digite 1, para parar, digite 0: ");
        scanf("%d", &seguir);
      }
    while (seguir);
    system("pause");
}
```

```
int calc_produto(int valor1, int valor2)
 {
    return valor1 * valor2;
 }
```

calc_produto: função definida pelo usuário (cont.)

- A função calc_produto é do tipo int, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor int.

- Para executar essa função é necessário fornecer dois parâmetros, os dois valores para cálculo do produto, oper1 e oper2.

Comando return():
retorno de valor e fim lógico da função

O comando return **atribui valor** a função.
Ao ser executado, **encerra a execução** da função.

Ao ser executado o return na função calc_produto, um valor é atribuído à função e ela encerra sua execução.

No ponto onde ocorreu a chamada de calc_produto, um valor passa a estar disponível para processamento.

Comando return():
retorno de valor da função (cont.)

Se uma função é declarada com tipo diferente de void (int, char, float, etc.) significa que ela pretende explorar a possibilidade de retorno de um valor em seu nome, e então pode ser usada em expressões.

Uma função que retorna um valor em seu nome deve conter pelo menos uma ocorrência do comando return, uma vez que é pela execução de um comando return que um valor é atribuído ao nome de uma função.

Quando uma função encerra sua execução?

Uma função encerra sua execução quando:

- o fim do seu código é atingido;

ou

- um comando *return* é encontrado e executado.

Vários comandos return podem existir em uma função?

Sim, embora não seja recomendável.

Segundo os princípios da programação estruturada seguidos na disciplina, cada função deve ter um único ponto de entrada e um único ponto de saída.

Se vários returns existirem em uma função, tem-se múltiplos pontos de saída possíveis.

Mas a função só conclui quando o primeiro *return* é ativado.

Funções em C

As funções devem ser declaradas de modo a serem o mais independentes possível do mundo externo a elas. Nos códigos das funções devem ser usados sempre que possível tão somente os parâmetros declarados no cabeçalho da função (se existirem) e os demais itens locais à função.

Em grande medida em C a preocupação com a independência das funções é facilitada pelo fato dos parâmetros de chamada e dos parâmetros da declaração da função ocuparem espaços de memória distintos e só existir a chamada passagem de parâmetro por valor entre eles.

Passagem de parâmetro por valor: os parâmetros de chamada e os parâmetros formais (da declaração da função) só se conectam no momento da chamada da função e então o que há é apenas a transferência de valores entre os parâmetros respectivos.

Exemplo de passagem de parâmetro por valor

A seguir duas versões de um programa que recebe um valor e através de uma função soma dez a este valor. Como a passagem de parâmetros para as funções em C é por valor, e o valor alterado não é mostrado no interior da função, a alteração do valor é perdida.

Observar nas duas versões do programa a seguir que como os mundos da função e o mundo externo a ela são mundos distintos, não faz diferença usar nomes iguais (valor e valor) ou diferentes (valor e num) nos parâmetros correspondentes usados na *main* e na função.

Exemplo de passagem de parâmetro por valor (cont.)

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
main (  )
{
    int valor;
    system("color 71");
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n",
                                valor);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    system("pause");
}

void soma_dez_a_valor(int valor)
 {
    valor = valor + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", valor);
 }
```

Exemplo de passagem de parâmetro por valor (cont.)

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
main (  )
{
    int valor;
    system("color 71");
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n",
                                valor);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    system("pause");
}

void soma_dez_a_valor(int num)
 {
    num = num + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", num);
 }
```

Exemplo de passagem de parâmetro por valor (cont.)


```
Valor inteiro: 34

Na Main: valor antes da chamada da funcao: 34

Na Funcao: valor dentro da funcao: 44

Na Main: valor apos chamada da funcao: 34
Pressione qualquer tecla para continuar. . .
```

PARÂMETROS

Reforçando:

Os nomes das variáveis declaradas no cabeçalho de uma função são independentes dos nomes das variáveis usadas para chamar a mesma função.

As declarações de uma função são locais a essa função. Os parâmetros declarados no cabeçalho de uma função existem somente dentro da função onde estão declarados.

PASSAGEM DE PARÂMETROS

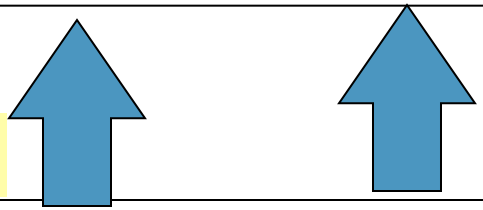
Ao ser ativada a função calc_produto, valor1 e valor2 são criadas.

E os valores existentes nesse momento em oper1 e oper2 são transferidos para valor1 e valor2.

A conexão entre oper1 e valor1 e oper2 e valor2 só existe no momento que a função é ativada.

Fora o momento da ativação as funções calc_produto e *main* são mundos independentes.

```
int calc_produto(int valor1, int valor2)
```



Chamada na função main:

```
calc_produto(oper1, oper2);
```

Atenção:

valor1 e valor2 existem na função calc_produto.

oper1 e oper2 existem na função *main*.

Quaisquer modificações de valor1 e valor2 que aconteçam a partir da chamada de calc_produto só são conhecidas e percebidas dentro da função calc_produto.

O quê é necessário para usar-se uma função em Linguagem C?

A declaração da função.

A chamada da função.

Dependendo do caso, o protótipo da função.

Declaração da função: cabeçalho e corpo da função, com o código que produz o(s) resultado(s) esperado(s).

Se for função com tipo diferente de void, deve ter pelo menos um return, para atribuir valor à função.

Chamada da função: no ponto onde se deseja que a função seja executada, deve ser escrito o nome da função seguido de um par de parênteses, tendo no interior o nome dos parâmetros, se houver.

Protótipo: as funções têm que ser declaradas antes de serem usadas. Para deixar a função *main* em destaque, é melhor declarar as funções definidas pelo usuário após a *main*. Então, para funções, o sistema aceita que primeiro só se indique o tipo, nome da função e tipos dos parâmetros, se houver , ou seja, o protótipo da função, e depois em algum ponto do código adiante, se declare a função de forma completa.

```c
//calcula produtos de pares de valores informados
#include <stdio.h>
#include <stdlib.h>
int calc_produto(int, int);
main (  )
{
    int seguir;
    int oper1, oper2, produto;
    do
      {
        printf("\nOperando 1: ");
        scanf("%d", &oper1);
        printf("\nOperando 2: ");
        scanf("%d", &oper2);

        printf ("\nProduto = %d\n", calc_produto(oper1, oper2));
        printf("\nMais um valor, digite 1, para parar, digite 0: ");
        scanf("%d", &seguir);
      }
    while (seguir);
    system("pause");
}
int calc_produto(int valor1, int valor2)
 {
    return valor1 * valor2;
 }
```

Protótipo

Chamada da função

Declaração da função

Forma geral de declaração de um protótipo:

tipo_da_funcao nome_da_função (lista de tipos dos parâmetros);

- **tipo_da_funcao**: o tipo de valor retornado pela função.
- **nome_da_função**: nome da função conforme as regras do C.
- **lista de tipos dos parâmetros**: tipo de cada parâmetro, separados entre si por vírgulas.

Cuidados no uso de funções com parâmetros

Em funções com parâmetros, cuidar que o número e o tipo dos parâmetros sejam coincidentes no protótipo (se usado), na declaração e na chamada.
Em C, os parâmetros independentemente de seus nomes são emparelhados na declaração e chamada por ordem de declaração, da esquerda para a direita.
Ex.:

```
int calc_produto(int, int);

int calc_produto(int valor1, int valor2)

calc_produto(oper1, oper2)
```

Aninhamento de funções é possível?

Em C, é possível chamar uma função de dentro de outra função, mas não é possível declarar uma função dentro de outra função!

Exercício

Escreva o código de uma função que calcule o fatorial de um número informado como parâmetro

Escreva um programa que use esta função

```
#include<stdio.h>
#include <stdlib.h>
int fatorial(int);      //prototipo da funcao fatorial
main(){
        int N;
        printf ("Informe o numero: ");
        scanf("%d",&N);
        printf("fatorial: %d\n",fatorial(N));
        system("pause");
}

// declaracao da funcao fatorial
int fatorial(int n){
        int I,fat=1;
        for (I=1;I<=n;I++)
                fat=fat*I;
        return(fat);
}
```

Problema na função fatorial definida: é do tipo inteiro, o que limita muito a sua aplicabilidade, pois o maior número do **tipo inteiro** é relativamente pequeno.

Solução: definir a função como do **tipo double**

```
#include<stdio.h>
#include <stdlib.h>
double fatorial(int);
main(){
        int N;
        printf ("Informe o numero: ");
        scanf("%d",&N);
        printf("fatorial: %lf\n",fatorial(N));
        system("pause");
}
// declaracao da funcao fatorial
double fatorial(int n){
        int I;
        double fat=1.0;
        for (I=1;I<=n;I++)
                fat=fat*I;
        return(fat);
}
```

Exercício

Escreva o código de uma função que calcule a média aritmética de dois valores informados como parâmetros

Escreva um programa que use esta função

```
#include<stdio.h>
#include <stdlib.h>
float media(float, float);
main(){

        float v1,v2,m;
        printf ("Informe os numeros: ");
        scanf("%f %f",&v1,&v2);
        m=media(v1,v2);
        printf("a media dos numeros e': %.4f\n",m);
        system("pause");
}
// declaracao da funcao media
float media(float n1, float n2){
        return((n1+n2)/2);
}
```

Exercício

Escreva o código de uma função que conte quantas ocorrências de um determinado caractere existem em um string. Ela recebe como entrada 2 parâmetros:
- um string de caracteres e
- o caractere a ser pesquisado.

Escreva um programa que use esta função

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
int contaChar(char[],char);
main(){

       char texto[100],c;
       printf ("\n Informe uma string: ");
       gets (texto);
       printf ("\nInforme o caractere a ser contado: \n");
       scanf("%c",&c);
       printf("o caractere %c aparece %d vezes no texto
\n",c,contaChar(texto,c));
       system("pause");
}
int contaChar(char s[], char ch) {
    int i,cont=0;
    for (i=0;i<strlen(s);i++)
        if (s[i]==ch) cont=cont+1;
    return cont;
}
```

Refências

- Material compilado a partir das seguintes fontes
  - C completo e Total
  - http://en.wikipedia.org/wiki/Modularity
  - Aula01 – Técnicas de Programação II, ppt
  - Software Engineering COMP 201, ppt
  - http://www.inf.ufrgs.br/~alvares/INF01040/FuncoesCidaLuis.ppt, ppt
  - http://www.telecom.uff.br/~marcos