#### Funções e Procedimentos

**Introdução**

**Funções na Linguagem C**

Funções são estruturas que permitem organizar programas grandes em blocos, tornando-os mais fáceis de ler e entender. Além de estruturar melhor o programa, funções permitem o reaproveitamento (reuso) de códigos, evitando repetição de trechos de códigos já construídos e testados, minimizando erros e facilitando a manutenção.

Estrutura geral de uma função na linguagem C:

**<tipo\_de\_retorno> nome\_da\_função(lista\_de\_parâmetros)**

**{**

**<corpo\_da\_função>**

**return valor\_de\_retorno;**

**}**

Onde:

* **tipo\_de\_retorno**: tipo do valor que a função vai retornar (*defaut* é **int**).
* **lista\_de\_parametros**: são as entradas (insumos) da função. Declarados no formato:

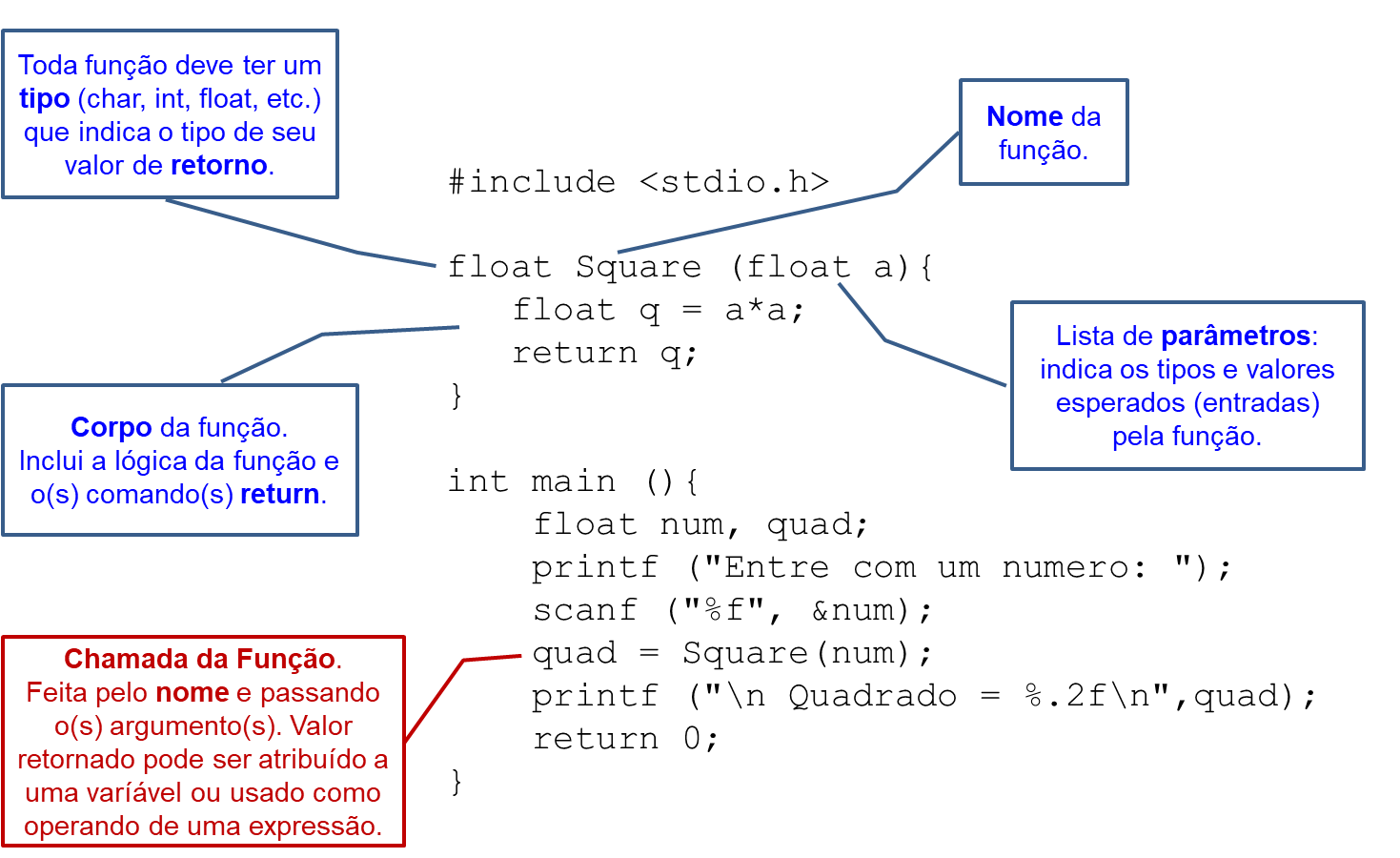
**(<tipo> arg1, <tipo> arg2, ..., <tipo> argN)**

* **corpo\_da\_função**: é o bloco de código – delimitado pelo { } onde é desenvolvida a lógica da função.
* **return**: comando que encerra execução da função e retorna valor do mesmo tipo de

**tipo\_de\_retorno**.

**Observação**: uma função com ***tipo de retorno*** definido deve obrigatoriamente ter uma instrução ***return*** em todos os caminhos possíveis de sua lógica, senão ocorre um erro em tempo de compilação.

**Exemplo:**



**Outro exemplo:**

#include <stdio.h>

int **EhPar** (int a) {

if (a%2 == 0) ***// Verifica se a e divisivel por dois***

**return** 1; ***// Retorna 1 se for divisivel***

else

**return** 0; ***// Retorna 0 se nao for divisivel***

}

int main (){

int num;

printf ("Entre com numero: ");

scanf ("%d",&num);

if(**EhPar**(num))

printf ("\n %d numero e´ par.\n", num);

else

printf ("\n %d numero e´ impar.\n", num);

return 0;

}

**Protótipo da função:**

Nada mais é do que a linha de cabeçalho (*header*) da função. Sua forma geral de declaração é:

<tipo\_de\_retorno> nome\_da\_função(lista\_de\_parâmetros);

O protótipo da função tem todas as informações sobre a *interface* da função, isto é, o seu nome, seus parâmetros de entrada e seu tipo de retorno. Deste modo, ao declarar um protótipo de função antes da função chamadora, já se informas ao compilador tudo o que é necessário para que a compilação possa ser bem sucedida.

A declaração da função propriamente dita (seu corpo, que detalha sua lógica de implementação) pode ser feita mais adiante, ou até mesmo em outro arquivo que pode ser compilado e testado separadamente do programa chamador. Esta característica é bastante útil especialmente no desenvolvimento de programas muito grandes e/ou complexos, portanto é boa prática de programação adotá-la na construção de programas em C.

**Exemplos**:

#include <stdio.h>

**float Square (float a); *// aqui declarado o protótipo da função***

int main (){

float num, quad;

printf ("Entre com um nro: ");

scanf ("%f", &num);

quad = Square(num);

printf ("\n Quadrado =

%.2f\n",quad);

return 0;

}

**float Square (float a){ *// aqui declarado o corpo da função***

**float q = a\*a;**

**return q;**

**}**

#include <stdio.h>

**int EhPar (int a); *// aqui declarado o protótipo da função***

int main (){

int num;

printf ("Entre com numero: ");

scanf ("%d",&num);

if(EhPar(num))

printf ("\n É par.\n");

else

printf ("\n É impar.\n");

return 0;

}

**int EhPar (int a){ *// aqui declarado o corpo da função***

**if (a%2)**

**return 0;**

**else**

**return 1;**

**}**

**Escopo de variáveis**

O escopo de uma variável é definido na linguagem C pela região – ou bloco de código, delimitado por { } – onde a variável é declarada. Cada variável é válida dentro do bloco onde é declarada.

Exemplo:



Nota-se pelo exemplo anterior que cada variável tem escopo no bloco onde foi declarada: a variável ***i*** tem escopo apenas no bloco do comando ***for***. As variáveis ***n*** e ***res*** têm escopo no corpo da função ***fat***. E as variáveis ***n*** e ***res*** declaradas no corpo do ***main*** têm escopo nesta região de código. É por isso que mesmo tendo os mesmo nomes, ***n*** e ***res*** são variáveis distintas em ***main*** e em ***fat*** por estarem em escopos distintos.

**Passagem de parâmetro por *valor*:**

O conteúdo (valor) do *argumento* é passado para o *parâmetro* respectivo.

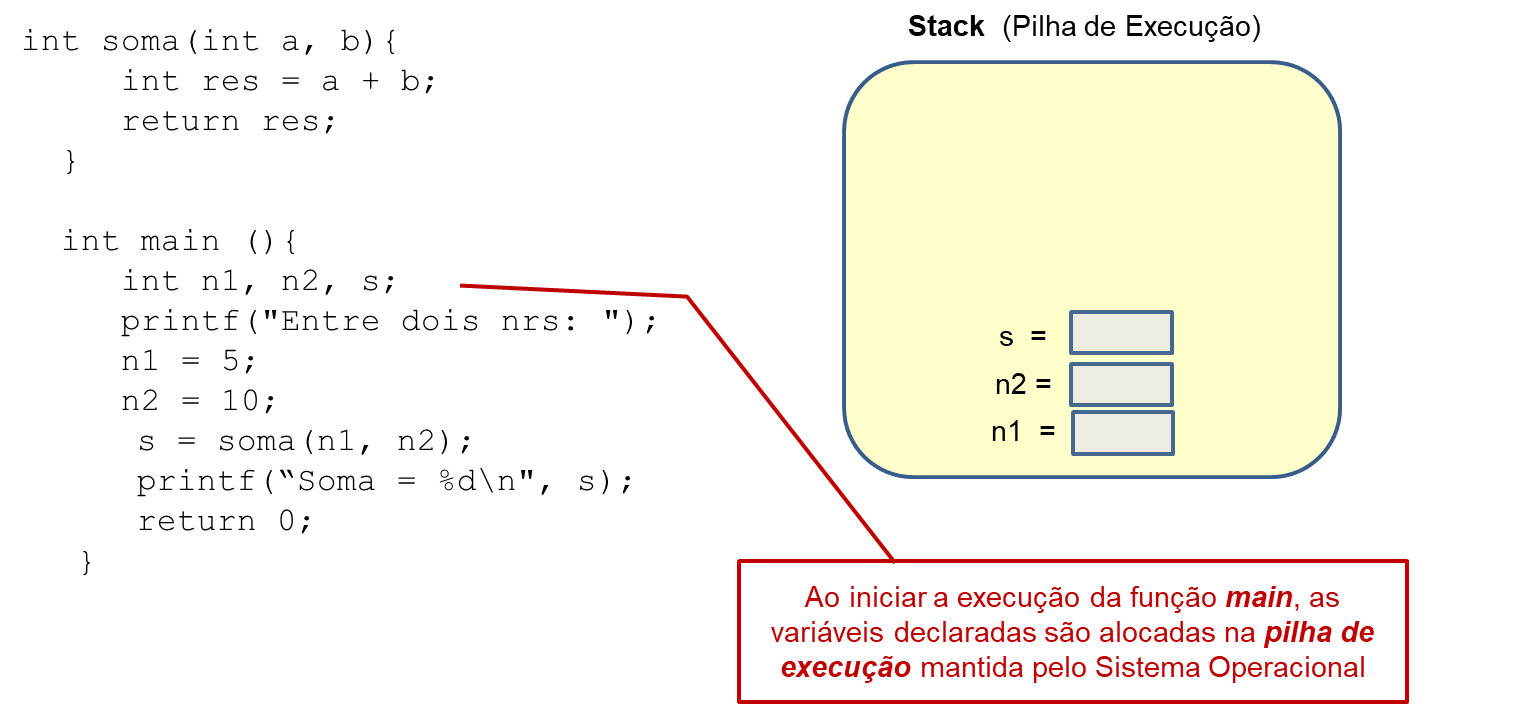
Exemplo:

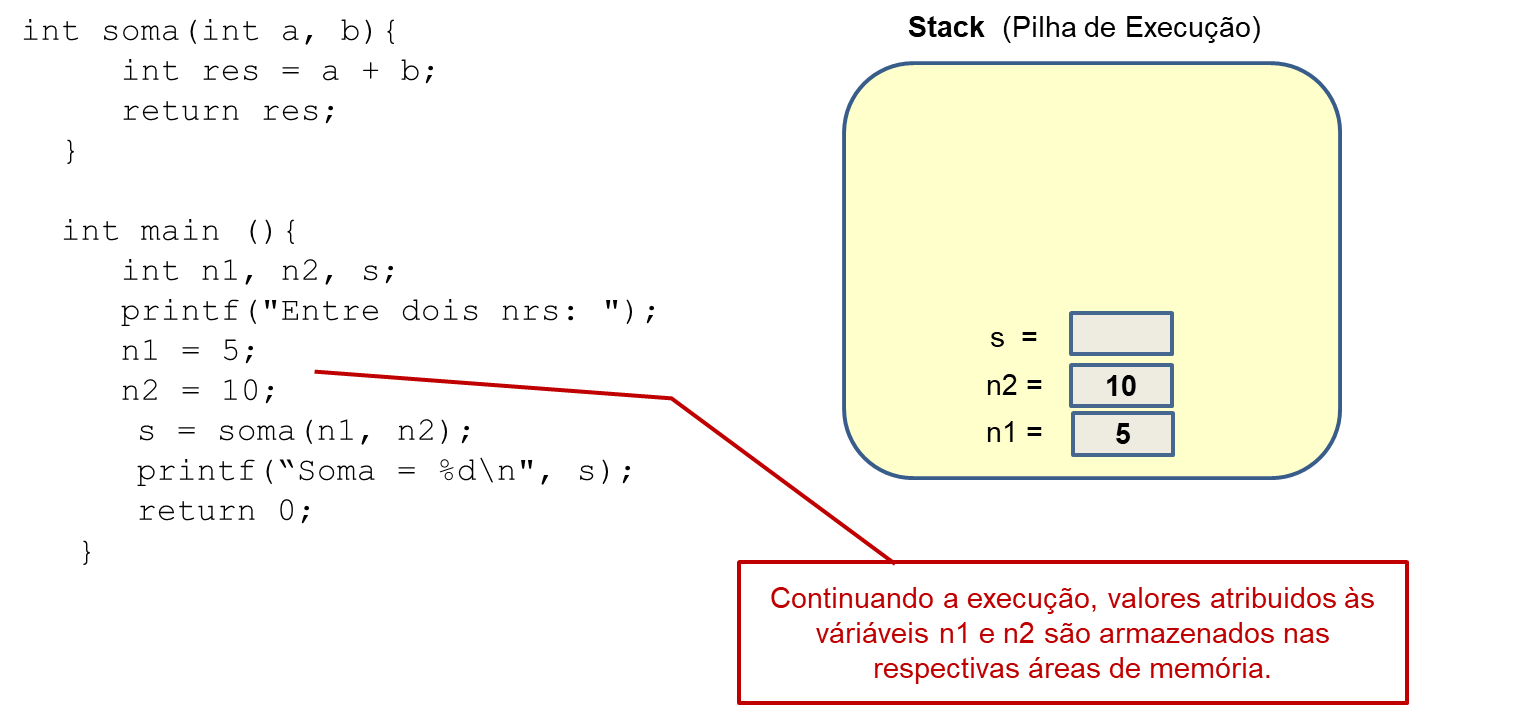
int soma(int a, b){  
 int res = a + b;  
 return res;  
}  
  
int main (){  
 int n1, n2, s;  
 n1 = 5;   
 n2 = 10;

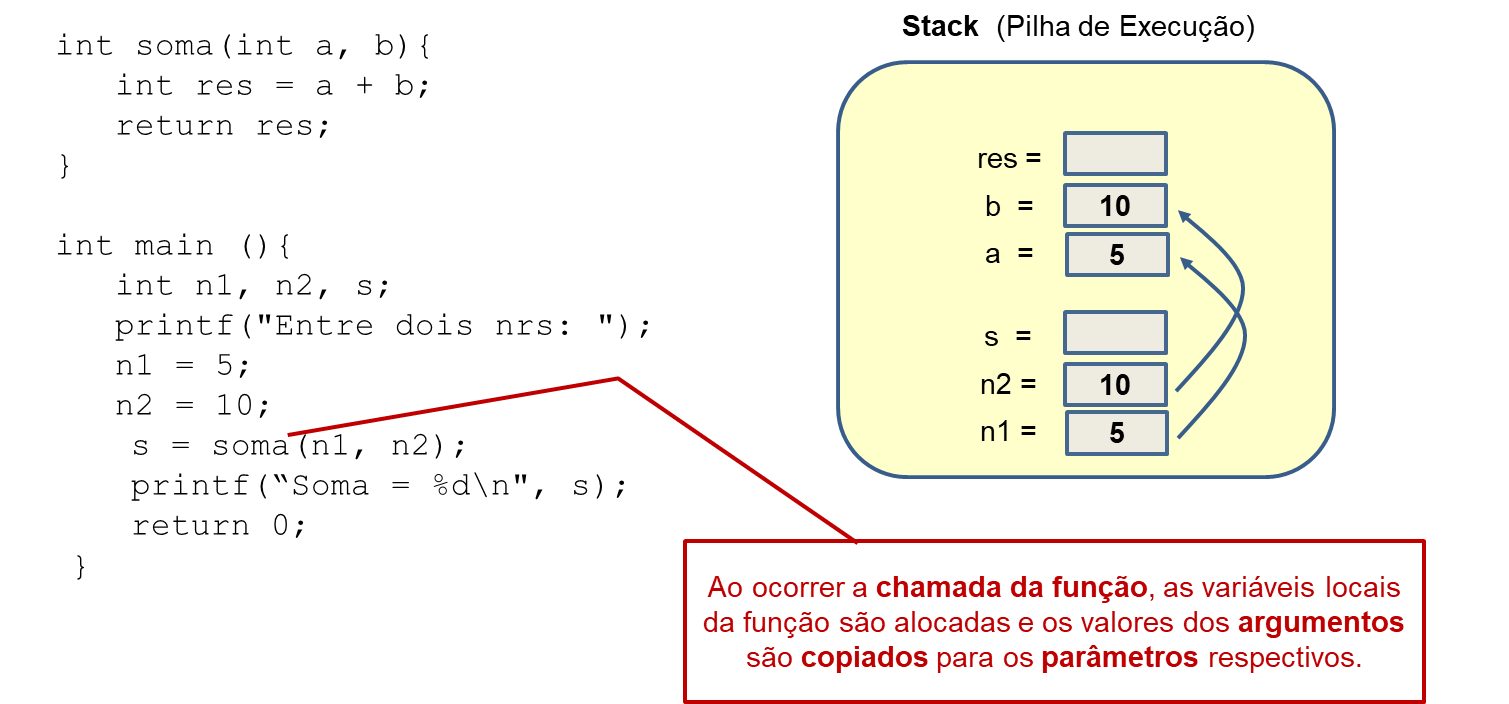
s = soma(n1, n2);  
 printf(“Soma = %d\n", s);  
 return 0;  
 }

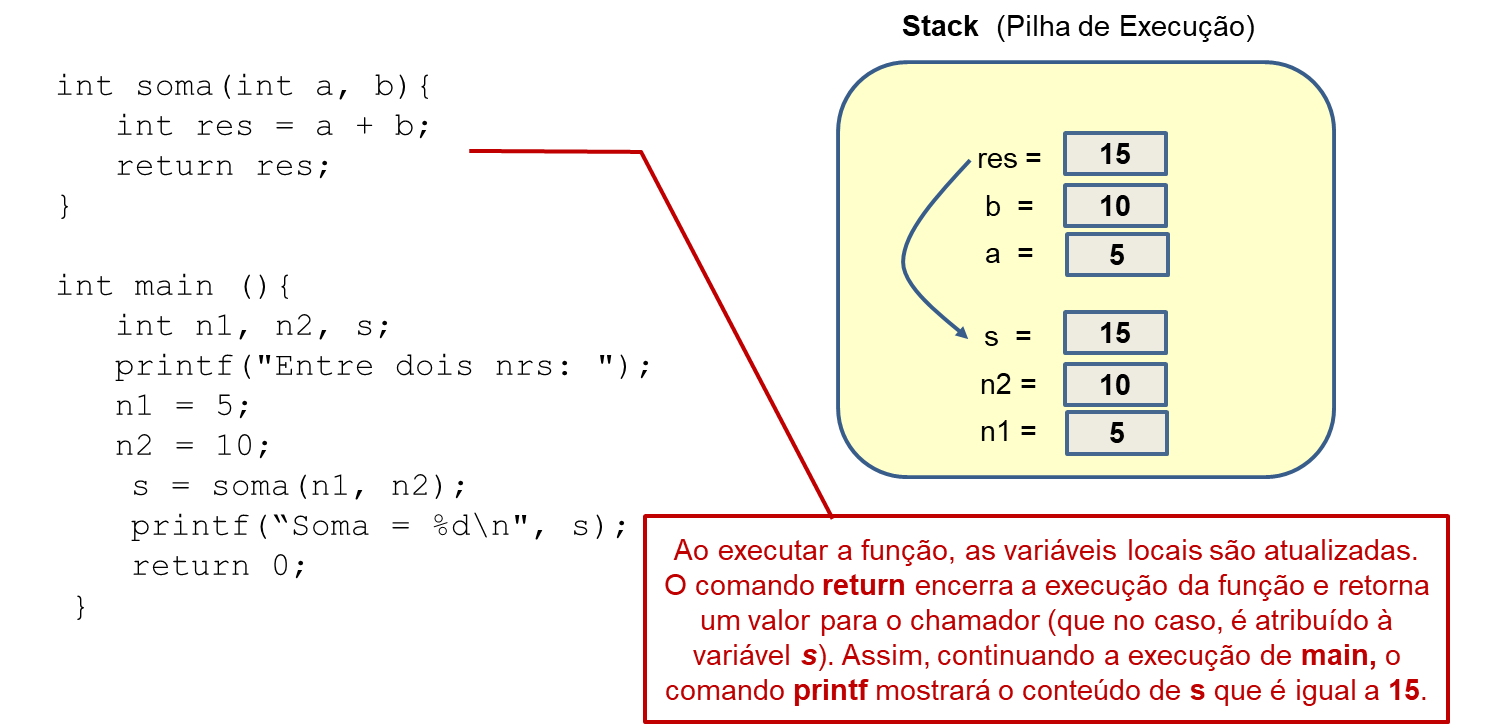
No exemplo, os valores das variáveis ***n1*** e ***n2*** são passados respectivamente para os parâmetros ***a*** e ***b*** da função ***soma***. Deste modo, a função calcula e retorna o valor 15, que é o valor atribuído para a variável ***s*** e mostrado no comando ***printf*** da função ***main***.[[1]](#footnote-1)

A seguir é detalhado o que acontece na memória do computador ao executar o programa acima, passo a passo.



****

****

****

**Referências (ponteiros) na linguagem C**

Antes de ir à passagem de parâmetro por referência, é necessário relembrar o conceito de ***ponteiro***. ***Ponteiro*** é uma variável que armazena um endereço de memória, em vez de um valor de um determinado tipo de dados. Na linguagem C, o operador & retorna o “endereço de”.

Exemplo: ao declarar uma variável qualquer:

int num = 10;

É alocada na memória uma área conhecida pelo programador pelo nome ***num***, mas que tem um endereço na memória, na ilustração o endereço de ***num*** é por hipótese ***xpto***.

****

Então, a instrução...

&num;

... retorna o endereço de *num*, ou seja, ***xpto***.

Outro operador importante em C é o operador \* , que indica que uma variável é um ***ponteiro*** (ou *apontador*, ou *pointer*), isto é, uma variável que armazena um endereço de memória e não um valor.

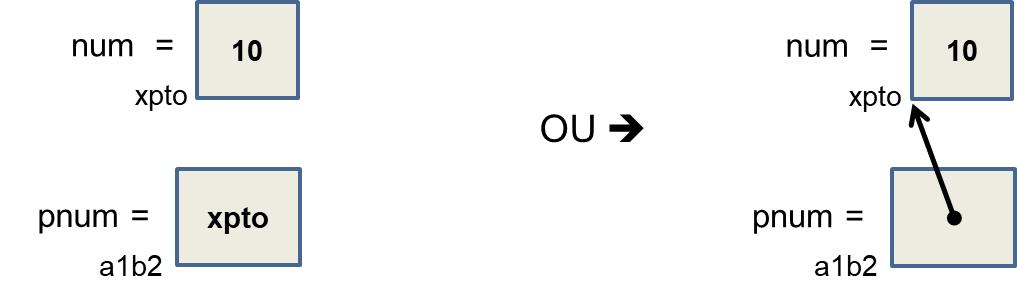
Exemplo:

int num; ***// define uma variável inteira***

int \*pnum; ***// define um ponteiro para inteiros***

num = 10; ***// atribui o valor 10 para num***

pnum = &num; ***// atribui o endereço de num para pnum***



Outro significado do operador \* , é de operador unário de referência, isto é, ele retorna o valor (conteúdo) de um endereço armazenado na variável ponteiro .

Exemplo:

int num;

int \*pnum;

num = 10;

pnum = &num;

printf(“Endereco de num eh %d e o valor de num eh %d",**pnum**,**\*pnum**);

O comando *printf* acima vai produzir a saída:

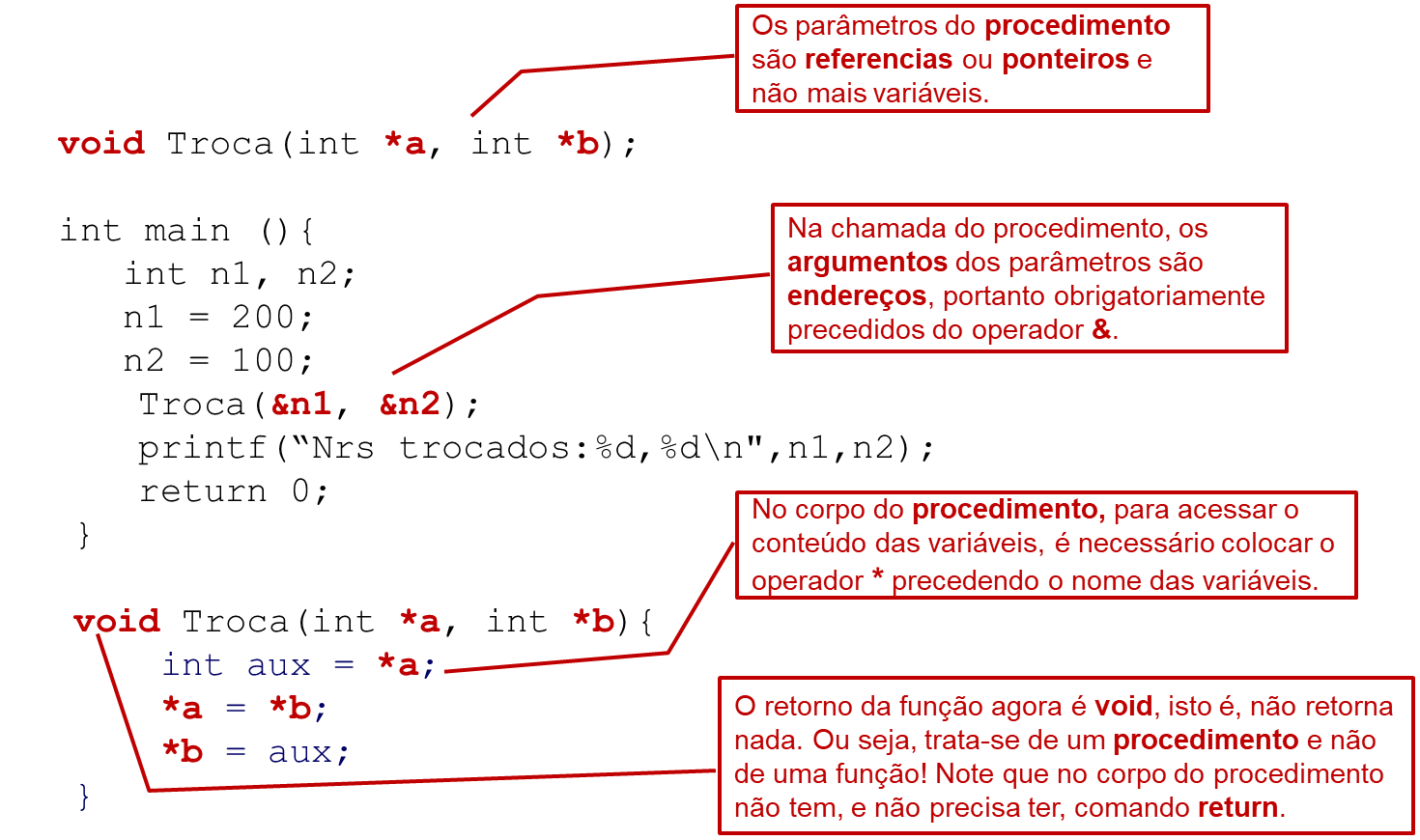
Endereço de num eh **xpto** e o valor de num eh **10**.

**Passagem de parâmetro por *referência*:**

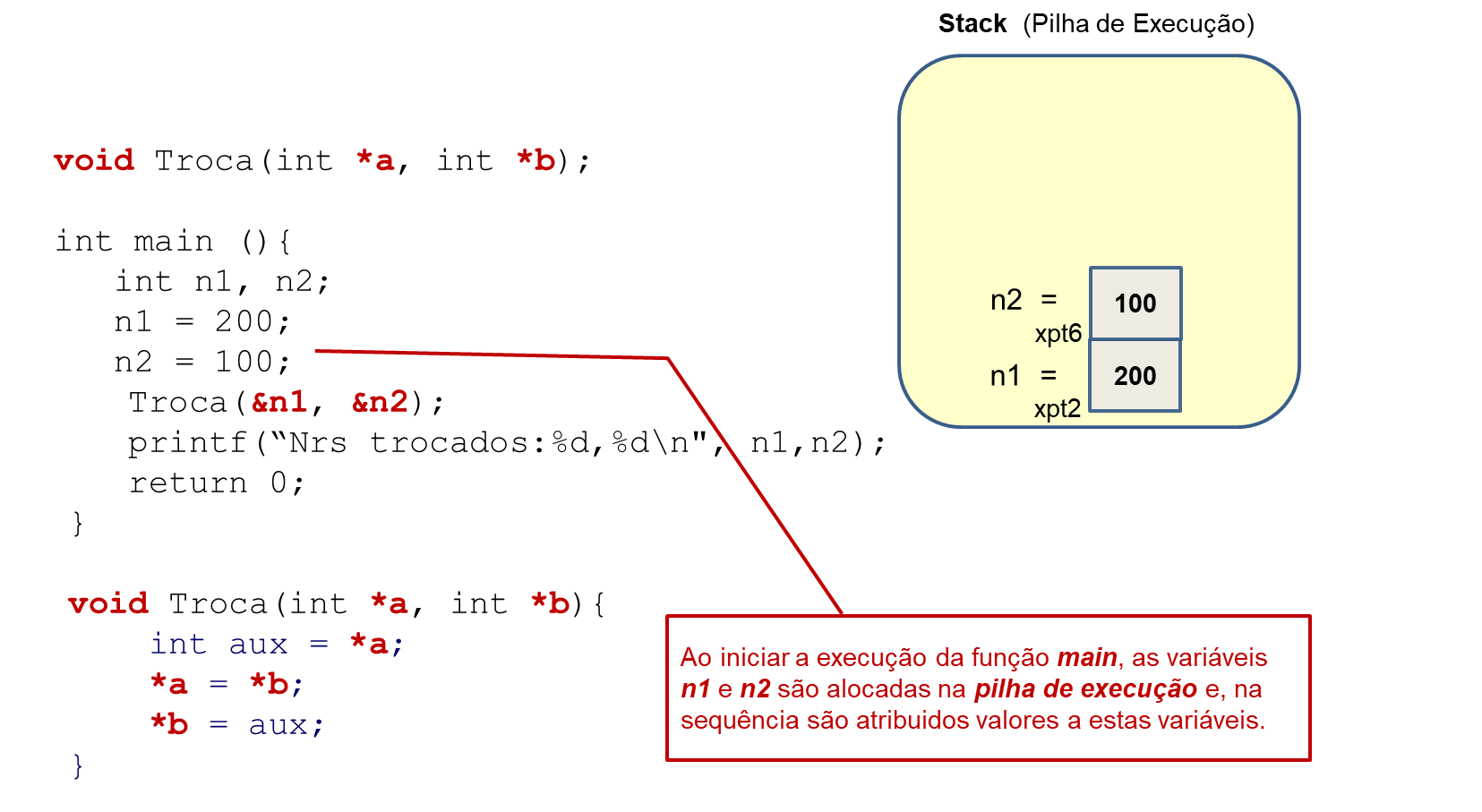
Sempre que possível, é recomendável utilizar forma de passagem de parâmetro por *valor*, para evitar “efeitos colaterais”, isto é, modificações indesejáveis nas variáveis passadas como argumento. Mas há situações onde deseja-se que estas modificações de fato ocorram. Por exemplo, quando quer-se que a função modifique (e retorne) mais de um valor. Como sabemos, por definição uma função retorna apenas um único valor!

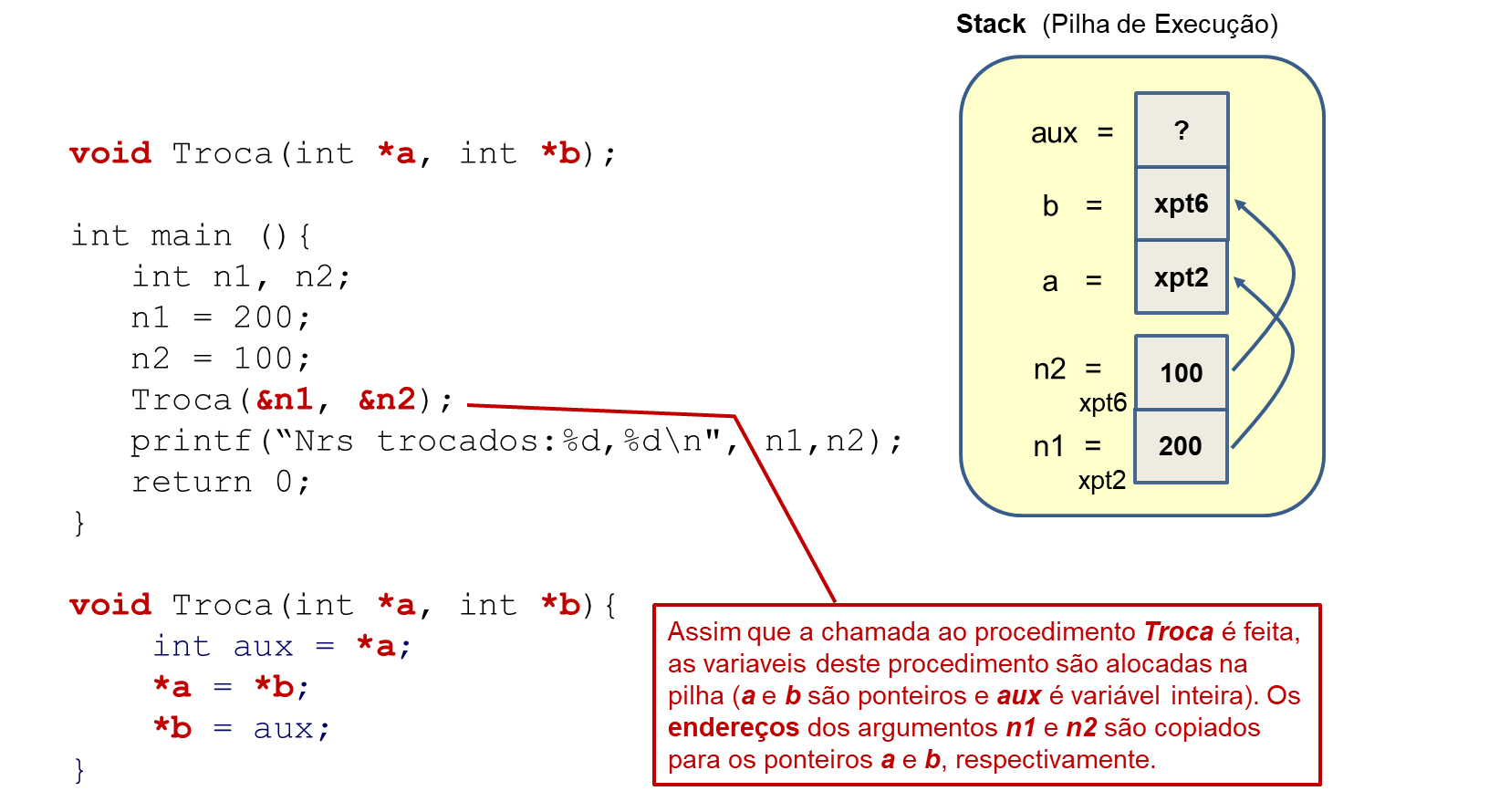
No caso de necessidade de modificar mais de um valor, pode-se declarar uma função com os *parâmetros* sendo ponteiros, isto é, referências. Os *argumentos* na chamada serão então *endereços de memoria* e não mais conteúdos (valores) a serem copiados.

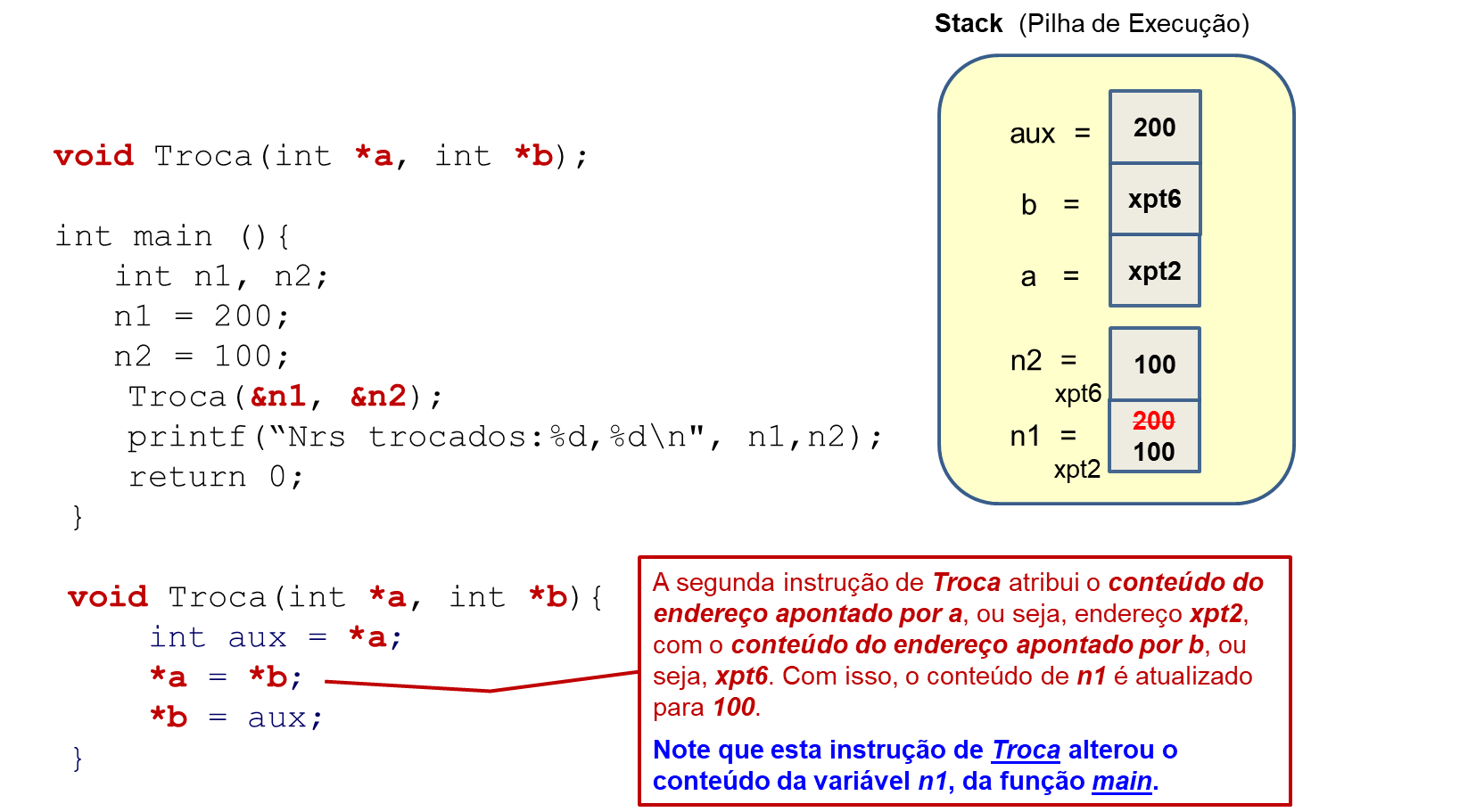
Exemplo: contruir uma função que receba como entrada dois valores inteiros e faça a troca destes valores e retorne os valores trocados. A seguir a solução, destacando-se as diferenças:

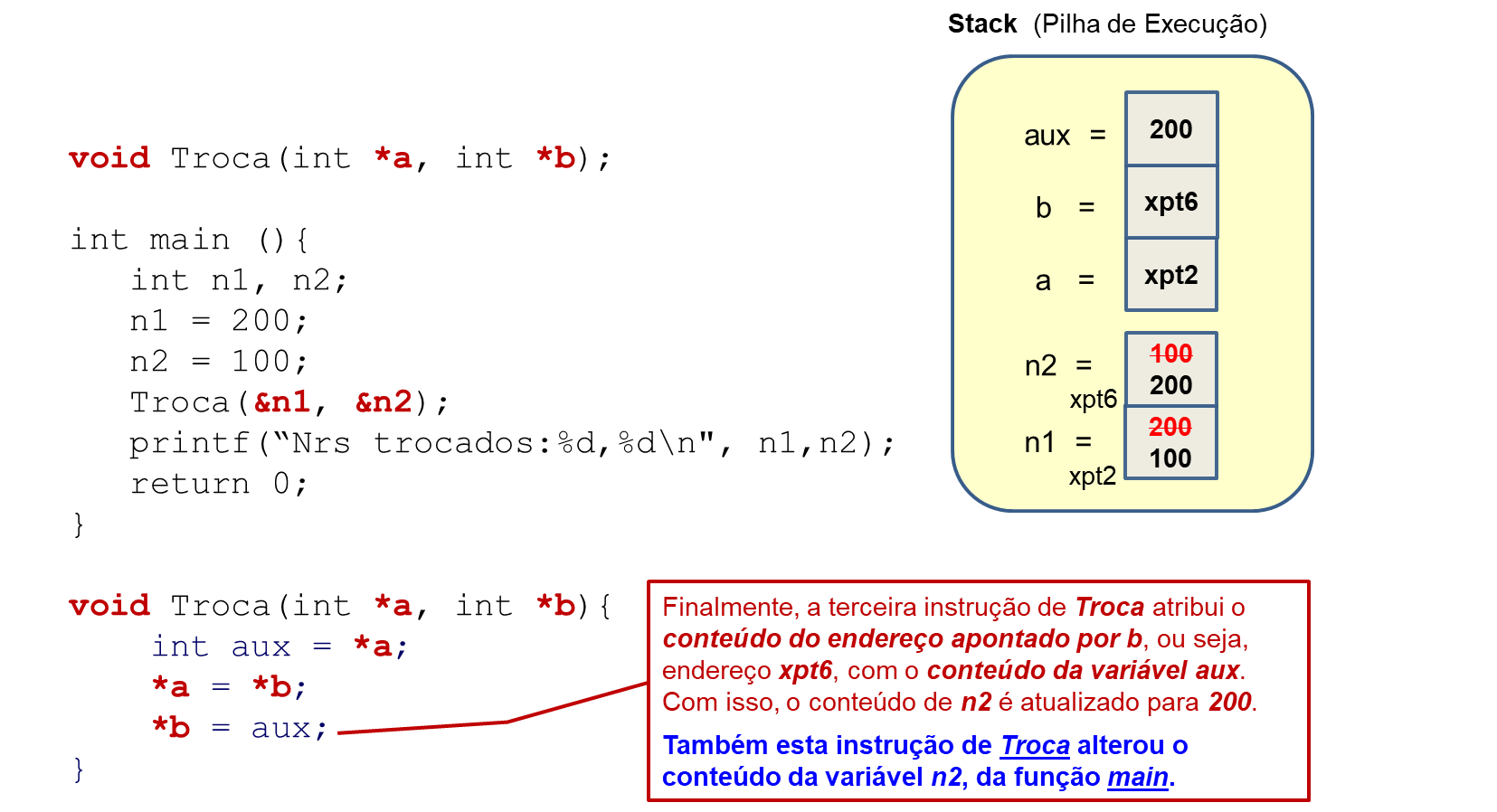
****

E o que acontece durante a execução do programa...









Exemplo:

Programa em C com um *procedimento* que receba dois números e calcule a soma e o produto destes números. A função ***main*** deve ler os valores pelo teclado, chamar o procedimento e imprimir os resultados.

void Calcula(int a, int b, int \*s, int \*p);

int main (){  
 int n1, n2, soma, prod;  
 printf(“Informe dois numeros inteiros: );  
 scanf(“%d”, &n1);

scanf(“%d”, &n2);

Calcula(n1, n2, &soma, &prod);  
 printf(“%d + %d = %d\n”, n1,n2, soma);

printf(“%d x %d = %d\n”, n1,n2, prod);  
 return 0;  
 }

void Calcula(int a, int b, int \*s, int \*p){

\*s = a + b;

\*p = a \* b;

}

1. Embora não haja concenso entre autores sobre essas nomenclaturas, no contexto desta apostila e desta disciplina será chamado ***parâmetro*** o valor definido como entrada da função (no exemplo em questão os ***parâmetros*** são ***a*** e ***b***) e ***argumento*** é a variável utilizada na chamada da função (no exemplo, os ***argumentos*** são ***n1*** e ***n2***). [↑](#footnote-ref-1)