



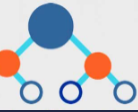
Estruturas de Dados 1

02 – Programação Estruturada ou Modular

Procedimentos, Funções e passagem de parâmetros

Antonio Angelo de Souza Tartaglia

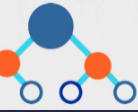
angelot@ifsp.edu.br



Estrutura de dados 1

Modularização ou programação estruturada?

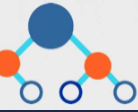
- A programação estruturada também é conhecida como programação modular;
- Para a utilização desta técnica, o desenvolvimento dos programas deve ser feito de forma descendente, com a decomposição do problema inicial em módulos ou estruturas hierárquicas, dividindo ações complexas em uma sequência de ações mais simples e desenvolvidas de forma mais fácil;



Estrutura de dados 1

Modularização ou programação estruturada?

- A programação estruturada enfatiza a utilização de unidades separadas de programas, chamadas de módulos, que são ativadas através de comandos especiais;
- Propõe que os programas sejam divididos em um conjunto de subprogramas menores. Cada um com seu objetivo específico e bem definido, mais fáceis de implementar e testar, seguindo a tática de “dividir para conquistar”.



Estrutura de dados 1

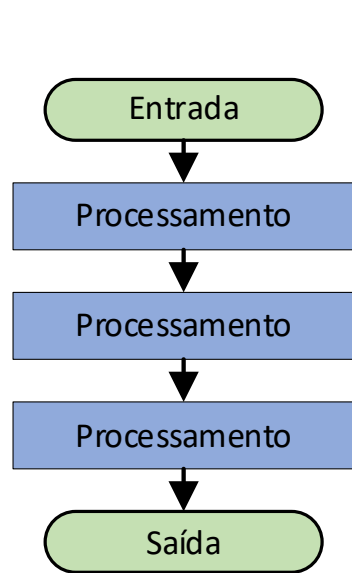
Modularização ou programação estruturada?

- Fundamenta-se no princípio básico de que um programa deve possuir um único ponto de entrada e um único ponto de saída, existindo de “1 a n” caminhos definidos desde o princípio até o fim do programa. Assim o programa deve ser composto por blocos elementares de instruções (comandos), interconectados através de três mecanismos de controle de fluxo de execução: sequência, seleção e iteração;
- Como outra característica da programação estruturada, temos que o uso de desvios incondicionais no programa, que sejam implementados com o comando goto (vá para), **estão totalmente proibidos.**

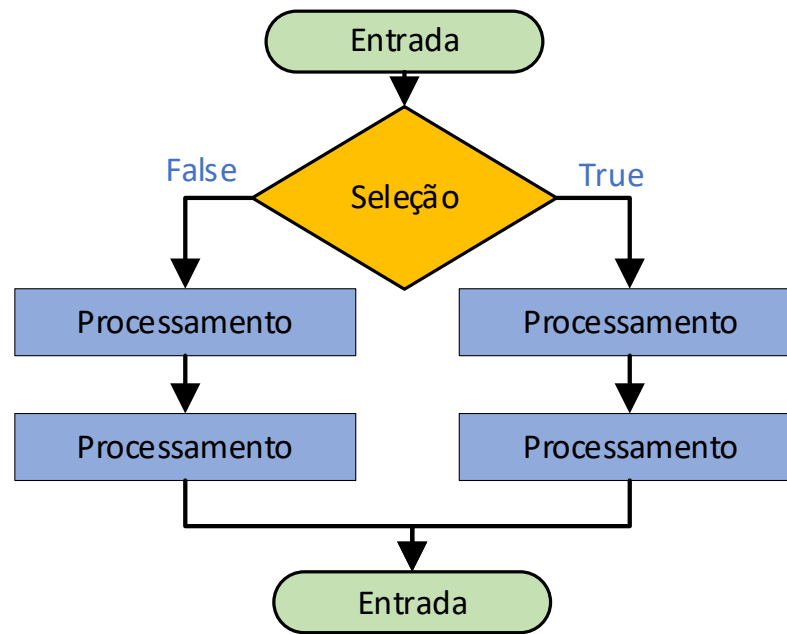


Estrutura de dados 1

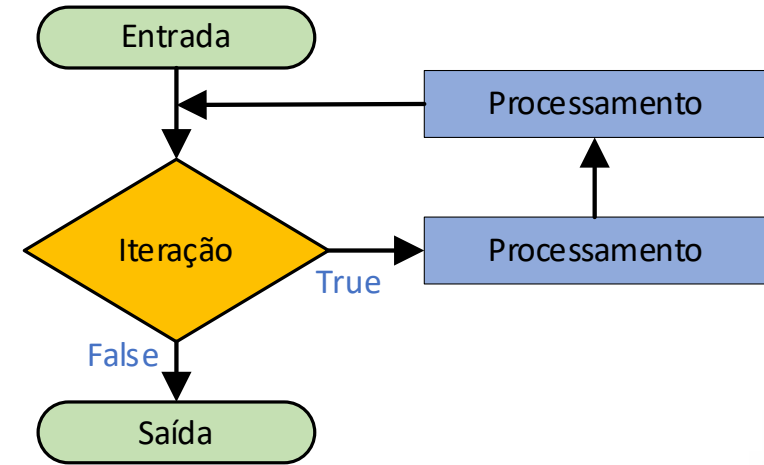
Modularização ou programação estruturada?



Sequencial



Seleção



Iteração

- Cada bloco elementar é delimitado por um ponto de início, que é necessariamente o início do bloco, e por um ponto de término, que é necessariamente o fim do bloco, ambos muito bem definidos.



Estrutura de dados 1

Modularização: procedimentos e funções ou Sub-rotinas

- Modularização consiste então, no processo de dividir o algoritmo em partes ou módulos, que chamamos de procedimentos ou funções, e que executam tarefas específicas.
 - Cada tarefa é representada por um módulo (procedimento ou função);
 - Um módulo é um grupo de comandos, constituindo um trecho de algoritmo;
 - O módulo deve ser bem definido, mais **independente** possível em relação ao restante do algoritmo e **executar uma tarefa apenas**, se possível.
- Dessa forma, procedimentos ou funções, nada mais são do que os blocos de códigos que podem ser nomeados e chamados de dentro de um programa. Em outras palavras, é uma sequência de comandos que recebe um nome e pode ser chamada de qualquer parte do programa, quantas vezes forem necessárias, durante a sua execução.



Estrutura de dados 1

Modularização: procedimentos e funções ou Sub-rotinas

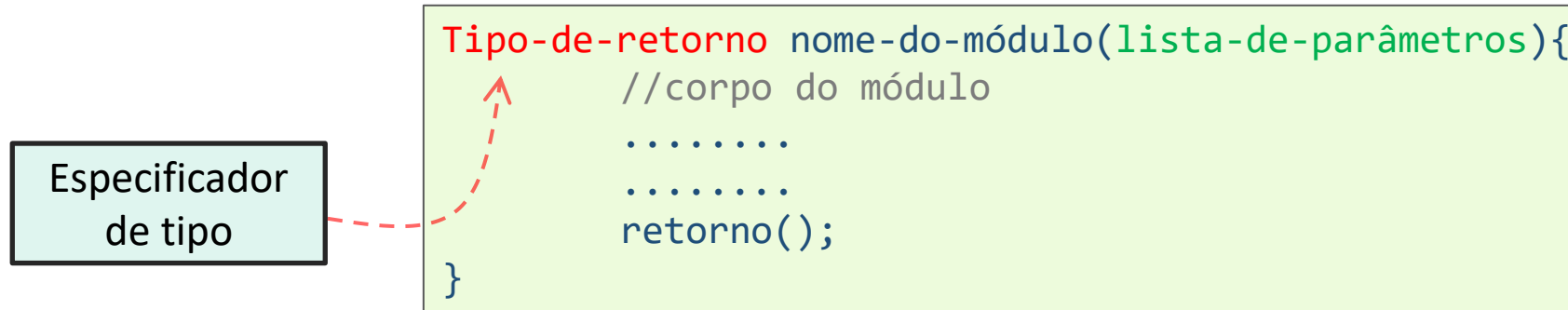
- Existem duas principais razões para o uso de módulos:
 - estruturação dos programas, e;
 - reutilização de código.
- Por estruturação dos programas podemos entender que agora, nossos programas serão construídos a partir de pequenos blocos de código (funções ou procedimentos), cada um deles com uma tarefa específica e bem definida. Isso facilita a organização, leitura e compreensão do programa.
- Assim, programas grandes e complexos são construídos bloco a bloco, por meio de seus módulos.



Estrutura de dados 1

Declaração dos Módulos

- Sintaxe:



- **Tipo de retorno**: retorno que o módulo devolverá. Ex.: se for uma função do tipo **float**, por exemplo, seu retorno também será **float**, se for do tipo **void**, não devolverá nada. Se nenhum tipo é especificado o compilador assume que a função devolve um resultado inteiro (**int**).
- **Nome do módulo**: Segue a mesma regra de nomeação das variáveis;
- **Lista de parâmetros**: Dados que são passados ao módulo através da declaração das variáveis que os receberão;
- **Corpo**: Instruções que realizam a operação pretendida; o algoritmo;
- **Retorno**: Valor que o módulo retorna, como resultado, para a parte do programa que faz a chamada.



Estrutura de dados 1

Local de Declaração dos Módulos ou Funções

- Um Módulo ou Função deve ser definido ou declarado antes de ser utilizado, e existem **dois locais possíveis**:
 - Entre as declarações de bibliotecas e a declaração da cláusula `main()`, na região onde são declarados os objetos globais, ou;
 - Ao final da função principal (`main()`), após a sua última chave de fechamento. Neste caso é necessária a declaração do **protótipo da função**, entre as declarações de bibliotecas e a declaração da cláusula `main()`;



Estrutura de dados 1

Local de Declaração dos Módulos ou Funções

- O protótipo de uma função, é uma declaração que **omite o corpo da função** mas especifica o seu nome, tipo de retorno e lista de parâmetros, desta forma:

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

```
int calculaTotal(int num1, int num2);

int main () {
    int x = 5, y = 2, resultado = 0;
    resultado = calculaTotal(x, y);
    printf("O Total é: %d", resultado);
    return 0;
}
```

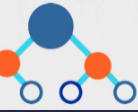
```
int calculaTotal(int num1, int num2) {
    return (num1 + num2);
}
```

Note que sua terminação é com ponto e virgula.

O comitê ANSI recomenda que seja feito o protótipo (modelo) de cada função, e esta é apenas uma declaração de um modelo para uma função que aparecerá em algum lugar dentro do programa.

Utilizando protótipos de todas as funções antes de chama-las, você informa ao compilador qual será a aparência da assinatura da função, qual o valor que ela retorna e a lista de argumentos, se houver.

Não é obrigatório incluir os nomes das variáveis, apenas seus tipos já são suficientes.



Estrutura de dados 1

Protótipos de função

- O padrão C ANSI expandiu a declaração tradicional de função para o ***protótipo de função***, permitindo que a quantidade, e o tipo de argumentos das funções sejam declarados.
- Os protótipos de função não faziam parte da linguagem C original. Porém, são um dos acréscimos mais importantes do comitê ANSI, à Linguagem C.
- Protótipos permitem que a linguagem forneça uma verificação mais forte de tipos. Quando protótipos são usados, o compilador pode encontrar e apresentar quaisquer conversões de tipos ilegais entre o argumento usado para chamar uma função e a definição de seus parâmetros. A linguagem também encontra diferenças entre o número de argumentos usados para chamar a função, e o número de parâmetros da função.



Estrutura de dados 1

Protótipos de função

- Dessa forma, o uso dos protótipos é **opcional**. Porém, eles habilitam o compilador a identificar qualquer incompatibilidade de tipos quando ocorre um erro, de forma que é uma boa ideia incluí-los.
- Protótipos de funções ajudam a detectar erros antes que eles ocorram. Além disso, eles auxiliam a verificar se seu programa está funcionando corretamente, não permitindo que funções sejam chamadas com argumentos inconsistentes.

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

int calculaTotal(int num1, int num2);

int main () {
    int x = 5, y = 2, resultado = 0;
    resultado = calculaTotal(x, y);
    printf("O Total é: %d", resultado);
    return 0;
}

int calculaTotal(int num1, int num2) {
    return (num1 + num2);
}
```



Estrutura de dados 1

Módulo ou Funções – Como funcionam

- Cada módulo pode definir as próprias estruturas de dados (variáveis, vetores, estruturas, etc.), suficientes e necessárias apenas para atingir o seu objetivo final;
- Um módulo é constituído por uma sequência de comandos que operam sobre um conjunto de objetos, que podem ser **globais** ou **locais**.

Neste caso, nossas variáveis.



Estrutura de dados 1

Variáveis Globais e Locais

- Variáveis que são declaradas dentro de uma função são chamadas de **variáveis locais**. Variáveis locais só podem ser referenciadas (utilizadas), por comandos que estão dentro do mesmo bloco no qual elas foram declaradas. Em outras palavras, variáveis locais não são reconhecidas fora de seu próprio bloco de código, fora do seu escopo, que é delimitado por **{ }**. Elas existem apenas enquanto o bloco de código em que foram declaradas está sendo executado, ou seja, são criadas na entrada de seu bloco de código e são destruídas na saída.

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

int calculaTotal(int num1, int num2);

int main () {
    int x = 5, y = 2, resultado = 0;
    resultado = calculaTotal(x, y);
    printf("O Total é: %d", resultado);
    return 0;
}

int calculaTotal(int num1, int num2) {
    int resultado;
    resultado = num1 + num2;
    return resultado;
}
```

Passagem por
cópia



Estrutura de dados 1

Variáveis Globais e Locais

- Ao contrário das variáveis locais, as **variáveis globais** são reconhecidas por todo o programa e podem ser usadas e alteradas por qualquer trecho de código. Além disso, elas guardam seus valores durante toda a execução do programa. Para sua criação, basta declará-las fora de qualquer função, na área destinada a declarações globais. Assim, elas podem ser acessadas por qualquer expressão independentemente de qual bloco de código contém a expressão.

Região de declarações
Globais

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

void calculaTotal(int num1, int num2);

int resultado;

int main () {
    int x = 5, y = 2;
    calculaTotal(x, y);
    printf("O Total é: %d", resultado);
    return 0;
}

void calculaTotal(int num1, int num2) {
    resultado = num1 + num2;
}
```

Lembre-se!! O main() também é uma função!



Estrutura de dados 1

Módulo ou Funções – Como funcionam

- Independentemente de onde uma função ou módulo seja declarado, seu funcionamento é basicamente o mesmo;
- O código do programa é executado até encontrar uma chamada de função. O programa é então **interrompido temporariamente**, e o fluxo de execução do programa passa para a função que foi chamada. Se houver parâmetros na função, os valores da chamada da função são **copiados para os parâmetros no código da função**;
- Os comandos da função são então executados;



Estrutura de dados 1

Funções ou Módulos

- Quando uma Função é chamada, o programa é pausado, e então passa a executar o corpo do módulo chamado;
- Após o fim da execução do módulo, o controle do programa retorna ao ponto onde ocorreu a chamada, em que estava pausado e continua sua execução;
- O programa principal pode chamar **qualquer** quantidade de funções;
- Uma função, ou módulo, pode chamar outras funções, inclusive a si própria;

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

```
void calculaTotal(int num1, int num2);
```

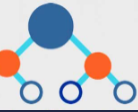
```
int resultado;
```

Exemplo de
variável global.

```
int main () {
    int x = 5, y = 2;
    calculaTotal(x, y);
    printf("O Total é: %d", resultado);
    return 0;
}
```

```
void calculaTotal(int num1, int num2) {
    resultado = num1 + num2;
}
```

Função utilizando a
variável global.



Estrutura de dados 1

Módulo ou Funções – Como funcionam

- Quando a função termina (seus comandos acabam ou o comando return foi encontrado), o programa volta ao ponto em que foi interrompido para continuar sua execução normal;
- Se houver um comando return, o valor após o comando será devolvido como resposta, e **copiado** para a variável que foi escolhida para receber o retorno da função.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int calculaTotal(int num1, int num2);
5
6  int main (){
7      int x = 5, y = 2, resultado = 0;
8      resultado = calculaTotal(x, y);
9      printf("O Total é: %d", resultado);
10     return 0;
11 }
12
13 int calculaTotal(int num1, int num2){
14     return (num1 + num2);
15 }
```

exemplo com variável local.



Estrutura de dados 1

Regras de escopo de funções

- São as regras que governam se um trecho de código tem “conhecimento” ou acesso a outra trecho de código ou dados.
- Em C, cada função é um bloco discreto de código. Um código de uma função é **privativo** àquela função, e não pode ser acessado por nenhum comando em outra função, exceto por meio de uma chamada à função. Por exemplo, não é possível utilizar o comando **goto** para saltar para o meio de outra função.
- O código que constitui o corpo da função é “escondido” do resto do programa e, a menos que se use variáveis ou dados globais, não pode acessar ou ser acessado, por outras partes do programa.
- As variáveis que são definidas internamente em uma função são chamadas de **variáveis locais**. Uma variável local de uma função passa a existir quando esta entra em execução, e são destruídas (usa posição de memória é liberada) no fim da execução da função.



Estrutura de dados 1

Argumentos de Funções ou Parâmetros Formais

- Parâmetros ou argumentos de uma função, são o que o programador utiliza para passar a informação de um trecho de código para dentro da função;
- Comportam-se como qualquer outra **variável local dentro da função**, e são **dinâmicos**. São **criados** na entrada e **destruídos** após o término da função;
- Basicamente, os parâmetros de uma função são uma lista de variáveis declaradas, separadas por vírgula, em que são especificados o tipo e o nome de cada variável que foi passada para a função;

```
int calculaTotal(tipo parâmetro1, tipo parâmetro2, ..., tipo parâmetroN){
```

```
    Blocos de comandos;
```

```
    ...
```

Deve-se assegurar de que os argumentos usados para chamar a função sejam compatíveis com o tipo de seus parâmetros, pois o compilador não gera mensagens de erro nestes casos, e assim, podem ocorrer resultados inesperados.



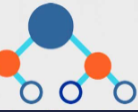
Estrutura de dados 1

Argumentos de Funções ou Parâmetros Formais

- Dependendo da função, ela pode não possuir nenhum parâmetro;
- Neste caso pode-se optar por duas formas:

```
void imprime(){  
    //corpo do módulo  
    .....  
    .....  
}
```

```
void imprime(void){  
    //corpo do módulo  
    .....  
    .....  
}
```

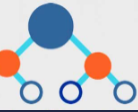


Estrutura de dados 1

Corpo do Módulo ou Função

- É no corpo de uma função, em seu bloco de comandos, que se define a tarefa que ela vai realizar quando for chamada;
- Em outras palavras, é no corpo da função que as entradas recebidas (parâmetros passados) são processadas, gerando a sua saída (return), ou outras ações são realizadas.

Uma função é construída com o objetivo de realizar uma única tarefa específica e bem definida, por exemplo: uma função para calcular a soma de dois valores deve ser construída de modo a receber estes determinados números como parâmetro, e retornar – **usando o comando return** – o valor calculado.



Estrutura de dados 1

O retorno da Função

- O retorno é a maneira como a função devolve o resultado da sua execução, para quem a chamou. Este valor, respeita o **tipo** estabelecido para a declaração da função que o gerou: este resultado sempre será do mesmo tipo ;

```
(int) calculaTotal(int num1, int num2);
```

- Uma Função pode ser e retornar qualquer tipo válido na Linguagem C: int, char, float, double, etc. **Inclusive, e principalmente, os tipos não primitivos que são criados pelo programador.**



Estrutura de dados 1

Procedimento: função que não retorna valores

- Procedimentos também são módulos de programas, capazes de executar uma tarefa;
- São definidos como tipo `void`, que é conhecido como tipo vazio.
- Uma função que seja declarada com o tipo `void` apenas executará o seu conjunto de comandos e, quando atingir o último comando em seu bloco (quando a chave de fechamento “`}`” for encontrada), sua execução é encerrada e **nenhum valor** será devolvido como resposta para o trecho de programa que a chamou.
- Dentro de um procedimento, assim como nas funções, podem ser utilizadas tanto variáveis locais quanto variáveis globais;
- Todas as variáveis locais pertencentes ao procedimento são **alocadas** (criadas) somente quando o procedimento entra em execução, e estas são **liberadas** (destruídas) quando o procedimento termina.



Estrutura de dados 1

Procedimentos na linguagem C

- Dessa forma, nos procedimentos, o **tipo** do resultado de retorno é `void` (perdido, vazio);
- No retorno do tipo **`void`** não é necessário utilizar a instrução `return`;
- `return` é usado apenas quando há a necessidade de abandonar o procedimento antes de seu término. Neste caso usa-se o comando `return` sem a expressão (valor que seria retornado). Sintaxe:

```
void nome-do-módulo(lista-de-parâmetros){  
    //corpo do módulo  
    .....  
    .....  
}
```

```
void nome-do-módulo(lista-de-parâmetros){  
    if(lista-de-parâmetros == erro)  
        return;  
    //corpo do módulo  
}
```



Estrutura de dados 1

Procedimentos na linguagem C

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

void imprime(int n);

int main (){

    imprime(10);
    system("pause");

    return 0;
}

void imprime(int n){
    int i;
    for(i = 0; i < n; i++){
        printf("\n\tLinha numero: %d", i + 1);
    }
    printf("\n\n\n");
}
```

```
"C:\Users\angelot\Desktop\Aulas 2\ se...
Linha numero: 1
Linha numero: 2
Linha numero: 3
Linha numero: 4
Linha numero: 5
Linha numero: 6
Linha numero: 7
Linha numero: 8
Linha numero: 9
Linha numero: 10
Pressione qualquer tecla para continuar. . .
```



Estrutura de dados 1

Procedimentos na linguagem C

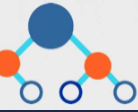
- Utilizando o comando **return** em procedimentos

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

```
void imprime_log(float x);
```

```
int main() {
    float f;
    printf("Digite f: ");
    scanf("%f", &f);
    imprime_log(f);
    system("pause");
    return 0;
}

void imprime_log(float x) {
    if(x <= 0) {
        printf("\n\n"); //salta linhas
        printf("Erro!! Valor tem que ser maior que 0!");
        printf("\n\n");
        return; //abandona a função
    }
    printf("\n\n\tLog de %.2f e: %.2f", x, log(x));
}
```



Estrutura de dados 1

Atividade 1

- Escreva um programa que implemente os dois módulos apresentados ao abaixo (cálculo de vantagens e deduções). As variáveis a serem utilizadas nos módulos, devem ser globais obrigatoriamente (**somente as que recebem atribuições dentro dos módulos**);
- O programa deve recolher os dados do usuário, armazená-los nas variáveis locais correspondentes no `main()`, e, em seguida passar os dados aos módulos, para que estes executem os cálculos;
- Imprima na tela os resultados obtidos pelos módulos. A rotina de impressão deve ficar fora dos módulos de cálculo, ou seja deve ficar no **`main()`**. Entregue no Moodle como Atividade 1.

```
Módulo calculoVantagens(parâmetros para cálculo)
    salarioBruto = numeroHoras * salarioHora;
    salarioFamilia = numeroFilhos * valorPorFilho;
    vantagens = salarioBruto + salarioFamilia;
fim calculoVantagens
```

```
Módulo calculoDeduções(parâmetros para cálculo)
    INSS = salarioBruto * 0,08;
    IRPF = salarioBruto * taxaIR;
    Deduções = INSS + IRPF;
fim calculoDeduções
```



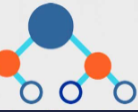
Estrutura de dados 1

Funções

- São módulos similares aos procedimentos, entretanto as funções retornam um valor após cada chamada;
- Uma função **não deverá simplesmente ser chamada**, como no caso dos procedimentos. Uma vez que a função retorna um valor, este deve ser atribuído à uma variável.

```
resultado = calculaTotal(x, y);
```

- Caso seja chamada e seu retorno não seja atribuído a nada, este valor é então descartado.



Estrutura de dados 1

Funções

- Sintaxe:

```
Tipo-de-retorno nome-do-módulo(lista-de-parâmetros){  
    //corpo do módulo  
    .....  
    .....  
    retorno (valor-de-retorno);  
}
```



Estrutura de dados 1

Funções

- É possível funções possuírem mais de um comando return:

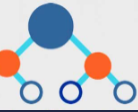
```
#include <stdio.h>
#include <stdlib.h>

int maior(int a, int b);

int main() {
    int x = 100, y = 200, resultado;
    resultado = maior(x, y);
    printf("\n\n\tMaior = %d \n\n", resultado);

    system("pause");
    return 0;
}
```

```
int maior(int a, int b) {
    if(a > b) {
        return a;
    } else {
        return b;
    }
}
```

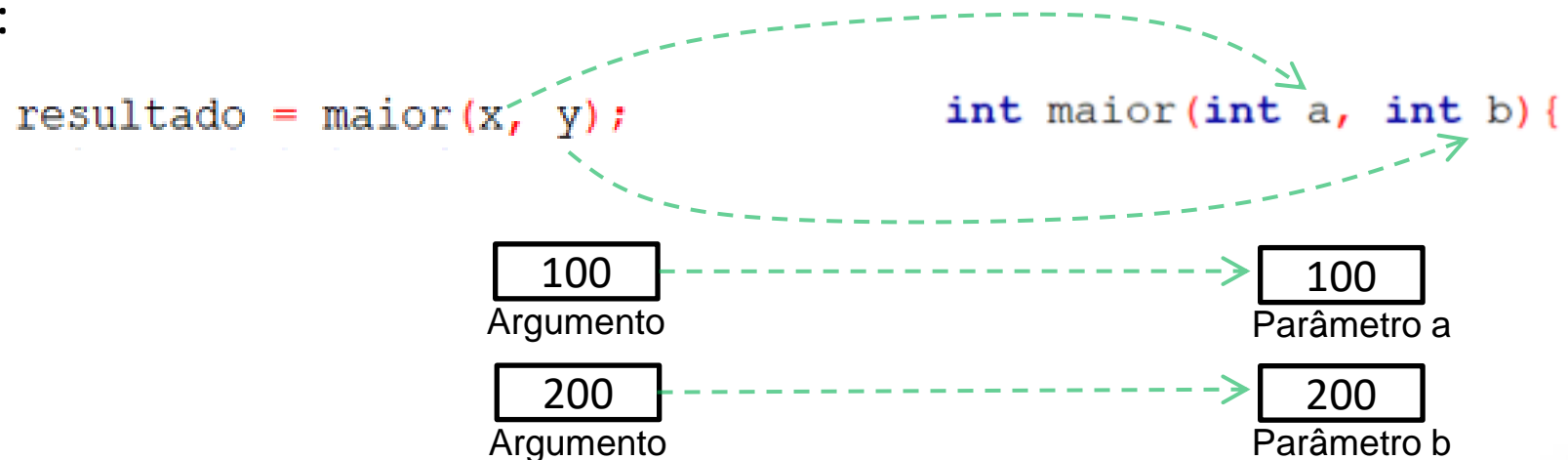


Estrutura de dados 1

Passagem por valor

- Esse método, que é o padrão, copia o valor de um argumento no parâmetro formal do módulo ou função.
- As alterações feitas nos parâmetros recebidos pelo módulo, não têm nenhum efeito nas variáveis utilizadas para efetuar a sua chamada.

- Exemplo:





Estrutura de dados 1

Passagem por valor

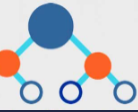
```
#include <stdio.h>
#include <stdlib.h>

int sqr(int x);

int main(void) {
    int num = 10;
    int num2 = sqr(num);
    printf("%d, %d \n\n\n", num, num2);
    system("pause");
}

int sqr(int x) {
    x = x * x;
    return(x);
}
```

```
C:\Users\angelot\Desktop\Aulas 1º seme...
10, 100
Pressione qualquer tecla para continuar. . .
```



Estrutura de dados 1

Passagem por referência (ponteiro)

- Nesse método, o endereço de um argumento é passado no parâmetro.
 - Dentro do módulo, o endereço é usado para acessar o argumento real utilizado na chamada;
 - Isso significa que alterações feitas no parâmetro **afetam a variável** usada para chamar a rotina;
 - A variável de referência é um apontador ou **ponteiro** (aponta para o endereço) para outra variável (variável referenciada).



Estrutura de dados 1

Passagem por referência

- Na passagem de parâmetros padrão, que é por valor, as funções não podem modificar o valor original de uma variável passada para a função, porque o valor é passado por **cópia**. Porém, existem casos em que é absolutamente necessário que toda modificação feita nos valores dos parâmetros dentro da função, seja passada de volta para quem a chamou. Principalmente quando desejamos **retornar mais de um valor como resposta** de uma função.
- Como um exemplo bastante simples disso temos a função `scanf()`:
 - Sempre que desejamos ler algo do teclado, passamos para a função `scanf()` o endereço da variável (“&”) onde o dado será armazenado. Essa variável então tem seu valor modificado através do acesso ao seu endereço, de dentro da função `scanf()`. Terminado o processo, seu valor pode ser acessado normalmente através do nome da própria variável, dentro do escopo ao qual esta pertence, programa principal ou funções que a utilizam.



Estrutura de dados 1

Passagem por referência – Ponteiros e endereços

- **Ponteiro**: tipo de variável que armazena **somente** os endereços de memória de **outras variáveis**;
- Endereço é a localização de uma variável na memória;
- Dizemos que o valor de um ponteiro “**aponta**” para uma variável (para o seu endereço) na memória, que dessa forma, pode ser acessada indiretamente com os operadores:
 - ***** acessa o conteúdo armazenado por uma variável cujo endereço está em um ponteiro. Ex.: *p, acessa o conteúdo **(do endereço)** que está guardado no ponteiro p;
 - **&** devolve o endereço de uma variável **(o endereço de)**.



Estrutura de dados 1

Passagem por referência

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

int sqr(int *x); //protótipo da função

int main(void) {

    int num = 10;
    int num2 = sqr(&num); // chamada da função

    printf("%d, %d \n\n\n", num, num2);
    system("pause");
}

int sqr(int *x) {
    *x = *x * *x;
    return(*x);
}
```

O & passa o endereço da variável num

```
Selecionar C:\Users\angelot\Desktop\Aula...
100, 100

Pressione qualquer tecla para continuar. . .
```



Estrutura de dados 1

Passagem por referência em um procedimento

- Exemplo em C:

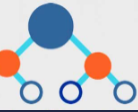
```
#include <stdio.h>
#include <stdlib.h>
```

```
void troca(int *x, int *y);
```

```
int main() {
    int a, b;
    printf("Digite dois valores: ");
    scanf ("%d %d", &a, &b);
    troca(&a, &b);
    printf("Conteúdo de a: %d - Conteúdo de b: %d \n\n\n", a, b);
    system("pause");
}
```

```
void troca(int *x, int *y) {
    //variável local
    int temp;

    temp = *x;
    *x = *y;
    *y = temp;
}
```



Estrutura de dados 1

Atividade 2

- O objetivo agora é substituir variáveis globais por ponteiros.
- Reescreva o programa da atividade 1, e agora não utilize mais variáveis globais, utilize somente variáveis locais e a passagem por referência nos dois módulos. Todas as variáveis devem ser declaradas dentro do **main()**, e somente as variáveis que receberão valores dentro das funções (que receberão atribuições, ou seja, o resultado dos cálculos), deverão ter seus endereços passados por referência. As demais variáveis que só entregam valores para cálculo, devem ter sua passagem de forma normal, ou seja, por cópia.
- Entregue no Moodle com atividade 2.

```
Módulo calculoVantagens(parâmetros para cálculo)

    salarioBruto = numeroHoras * salarioHora;
    salarioFamilia = numeroFilhos * valorPorFilho;
    vantagens = salarioBruto + salarioFamilia;

fim calculoVantagens
```

```
Módulo calculoDeduções(parâmetros para cálculo)

    INSS = salarioBruto * 0,08;
    IRPF = salarioBruto * taxaIR;
    Deduções = INSS + IPRF;

fim calculoDeduções
```