

AULA DE ALGORITMO - 05

Profa. M. Sc. Valéria Maria Volpe

Modularização de Programa

2

□ **Decomposição (Refinamento)**

- ▣ A decomposição de um problema é fator determinante para a redução da sua complexidade.
- ▣ Quando decompomos um problema em subproblemas estamos dividindo também a complexidade e simplificando a resolução.
- ▣ Outra grande vantagem da decomposição é que permite focalizar a atenção em um problema pequeno de cada vez, o que no final produzirá uma melhor compreensão do todo.

Modularização de Programa

3

□ **Decomposição (Refinamento)**

■ É conveniente adotarmos o seguinte processo de decomposição:

- Dividir o problema em suas partes principais;
- Analisar a divisão obtida para garantir a coerência;
- Se alguma parte ainda permanecer complexa, decompô-la também;
- Analisar o resultado para garantir o entendimento e a coerência.

Modularização de Programa

4

□ Módulo

- Depois de decompor um problema em subproblemas podemos construir **Subprogramas ou Módulos**.
- Portanto, um **Módulo** é a **divisão** do Programa em **programas menores** (subprogramas), sendo que cada **Módulo** faz uma **função específica** para resolver o problema que se deseja informatizar.

Modularização de Programa


5

□ Módulo

- ▣ Um Módulo é identificado pelo seu cabeçalho (também chamado de assinatura do módulo).

□ Sintaxe – Linguagem C

```
tipo de retorno NomeModulo (Lista de Parâmetros)
{
    Código do Módulo
}
```



Cabeçalho
(assinatura)
do Módulo

Modularização de Programa

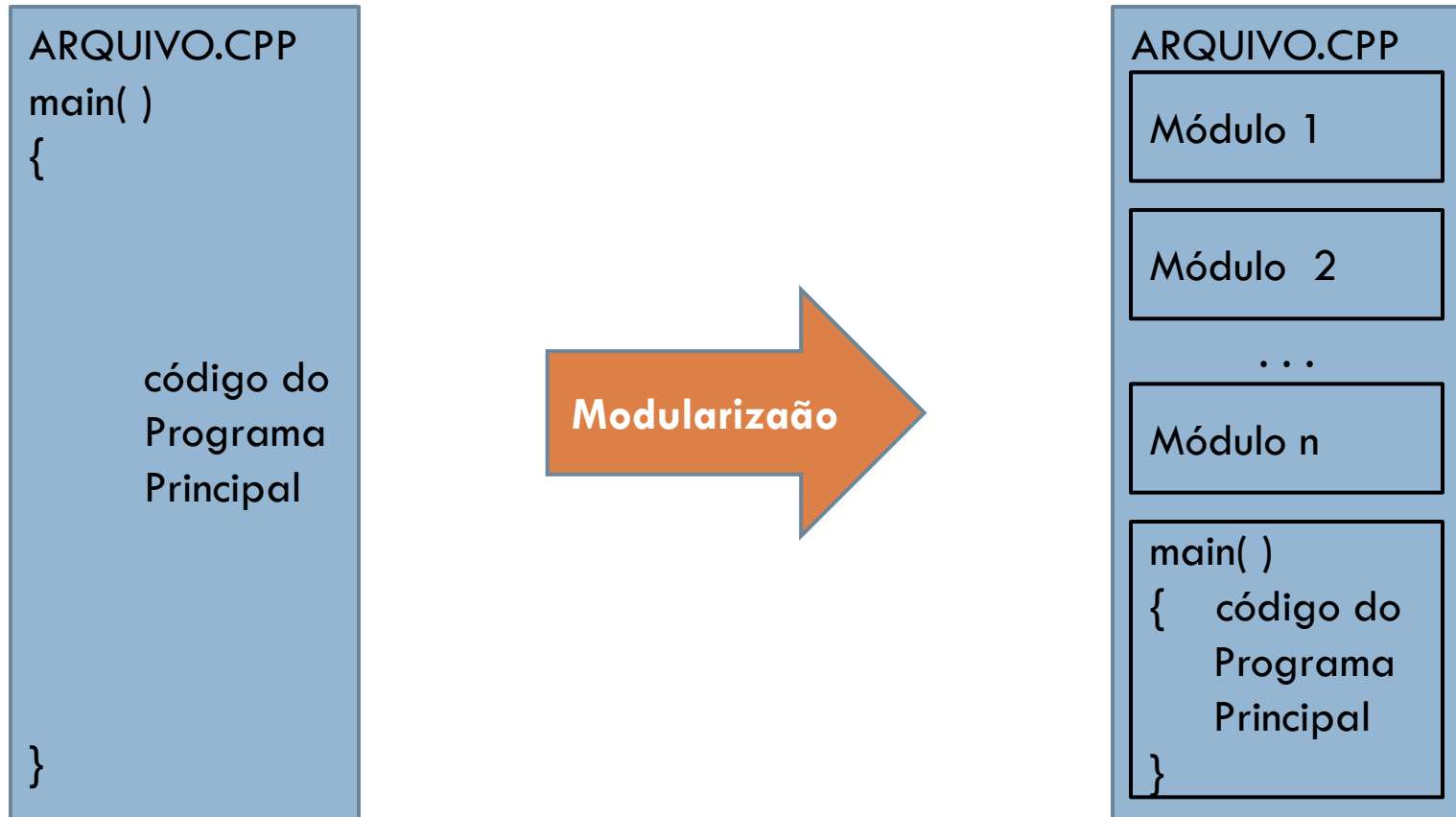
6

□ **Execução do Módulo**

- ▣ Um Módulo é executado toda vez que ele é “chamado”.
- ▣ A chamada ao Módulo é feita pelo seu nome.
Assim, ele executado.

Modularização de Programa

7



Modularização de Programas

8

□ Exercícios

1. Faça um programa modularizado para mostrar a mensagem: “Hello World My First Program!”

Solução Exercício 1

9

```
#include <stdio.h>
#include <conio.h>
// Módulo
void helloWorld()
{
    printf(" Hello World My first Program");
}
main()
{
    helloWorld(); // chamada para execução do módulo
    getch();
}
```

Modularização de Programa

10

□ Lista de Parâmetros

- ▣ A Lista de Parâmetros é um conjunto de variáveis, **declaradas** no cabeçalho do Módulo, que representa as informações que o Módulo **recebe** quando é chamado.

Modularização de Programa

11

□ Módulo que retorna Valor

- Um modulo pode, quando necessário, retornar **um único** valor (um n° inteiro, um n° real, um caracter), ou não retornar nenhum valor.
- Para o módulo dar retorno usa-se a palavra reservada **return**.
- Também é necessário indicar, no cabeçalho do Módulo o tipo do valor que ele retorna.
- A execução do **comando return finaliza** a execução do Módulo (Bloco de Comandos onde ele está inserido).

Modularização de Programa

12

□ Módulo que retorna Valor

Tipo de retorno	O que o Módulo irá retornar
int	Significa que o Módulo irá enviar (retornar) um número inteiro
float	Significa que o módulo irá enviar (retornar) um número real
char	Significa que o módulo irá enviar (retornar) um único caracter
void	Significa que o Módulo NÃO dá retorno

Modularização de Programa

13

□ Retorno do Módulo

- Quando um Módulo dá retorno é necessário que **uma variável** receba esse valor.
- Esta **variável** deve ser do **mesmo tipo** do valor retornado.
- A execução do Módulo ocorre sempre que aparecer seu nome.

Modularização de Programa

14

□ Exemplo:

```
int somarXY (int X, int Y)
```

```
{
```

```
    int SOMA;
```

```
    SOMA = X + Y;
```

```
    return (SOMA);
```

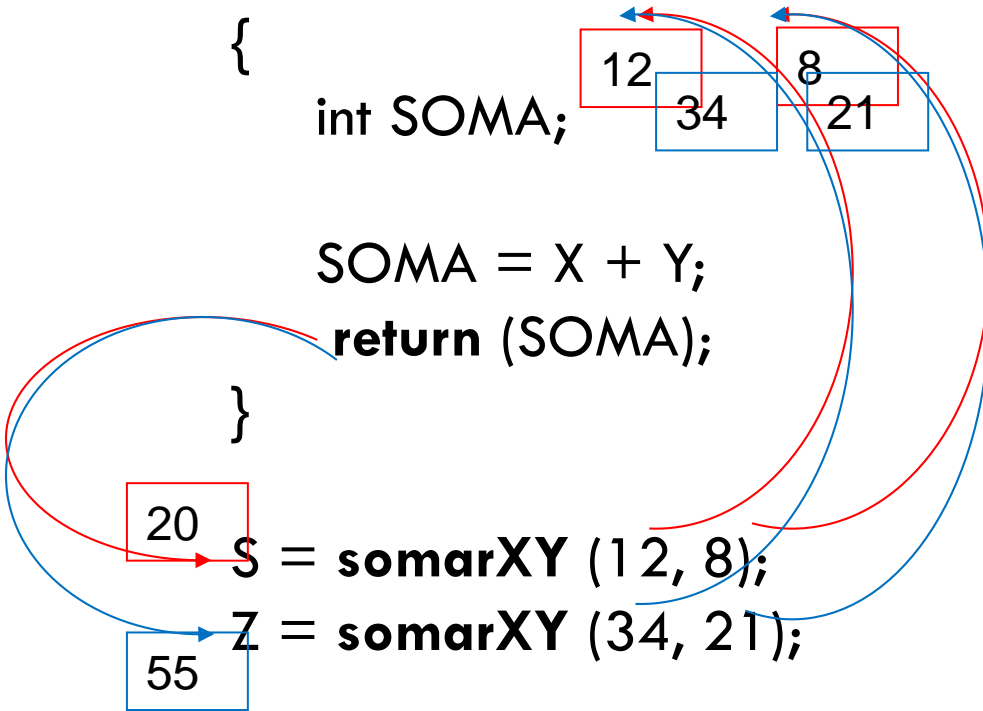
```
}
```

20

```
S = somarXY (12, 8);
```

55

```
Z = somarXY (34, 21);
```



Modularização de Programas

15

□ Exercícios

1. Faça um programa modularizado para Calcular o somatório:

$$\sum_{i=1}^n \frac{i+1}{i^2} = \frac{2}{1} + \frac{3}{4} + \frac{4}{9} + \frac{5}{16} + \dots + \frac{n+1}{n^2}$$

Solução Exercício 1

16

```
#include <stdio.h>
#include <conio.h>
// Módulo para ler um número inteiro
int lerNumero()
{
    int x;
    printf(" digite um número inteiro: ");
    scanf("%d", &x);
    return (x);
}
```

// Continua no próximo slide

Solução Exercício 1

17

// Módulo para calcular o somatório

```
float calcularSomatorio(int n)
```

```
{
```

```
    float somatorio = 0.0;
```

```
    int i;
```

```
    for(i = 1; i <= n; i = i + 1)
```

```
    {
```

```
        somatorio = somatorio + ((i + 1.0) / (i * i));
```

```
    }
```

```
    return (somatorio);
```

```
}
```

// Continua no próximo slide

Solução Exercício 1

18

// Módulo para mostrar os resultados

```
void mostrarResultado(float somatorio)
```

```
{
```

```
    printf("\n Somatório = %.2f", somatorio);
```

```
}
```

// Módulo principal (programa principal)

```
main()
```

```
{
```

```
    int n;
```

```
    float somatorio;
```

```
    n = lerNumero();
```

```
    somatorio = calcularSomatorio(n);
```

```
    mostrarResultado(somatorio);
```

```
    getch();
```

```
}
```

Exercício para resolver

19

- 1) Faça um **programa modularizado** que leia o peso, a altura e o sexo de uma pessoa, que calcule o peso ideal e verifique se a pessoa está acima, abaixo ou no peso ideal. Calcule o peso ideal de acordo com a fórmula abaixo:
 - ▣ Homem: $(72.2 * \text{altura}) - 58$
 - ▣ Mulher: $(62.1 * \text{altura}) - 44.7$
- 2) Faça um **programa modularizado** para ler 3 números reais e verificar se é possível formar um triângulo ou não. Se formar um triângulo verifique que tipo de triângulo é formado: equilátero (três lados iguais), isósceles (dois lados iguais e um diferente) ou escaleno (três lados diferentes). O programa deve repetir 10 vezes, para 10 possíveis triângulos. Sabendo que para formar um triângulo é necessário que:
 - ▣ $\text{Lado_A} < \text{Lado_B} + \text{Lado_C}$ E $\text{Lado_B} < \text{Lado_A} + \text{Lado_C}$ E $\text{Lado_C} < \text{Lado_A} + \text{Lado_B}$

Solução Exercício 1

20

```
#include <stdio.h>
#include <conio.h>
// Módulo para ler um caracter que representa o sexo
char lerSexo()
{
    char s;
    printf("\n Digite seu sexo F - Fem / M - Masc: ");
    s = getche();
    return s;
}
// Continua no próximo slide
```

Solução Exercício 1

21

// Módulo para ler um número real que representa a altura ou o peso

float lerPesoAltura()

{

 float pesoaltura;

 scanf("%f", &pesoaltura);

 return pesoaltura;

}

// Continua no próximo slide

Solução Exercício 1

22

```
// Módulo para calcular o peso ideal
float calcularPesoldeal(float altura, char sexo)
{
    switch(sexo)
    {
        case 'F': case 'f':
            return((62.1 * altura) - 44.7);
            break;
        case 'M': case 'm':
            return((72.2 * altura) - 58.0);
            break;
        default: return (0.0);
    }
}
```

// Continua no próximo slide

Solução Exercício 1

23

```
// Módulo para verificar se a pessoa está acima, abaixo ou no peso ideal
void verificarPesoIdeal(float peso, float pesoideal)
{
    if(peso == pesoideal)
    {
        printf("\n\n Parabéns você está no peso ideal: %f Kg \n\n", pesoideal);
    }
    else
    {
        if(peso > pesoideal)
        {
            printf("\n\n Você está acima do peso ideal: %f Kg \n\n", pesoideal);
        }
        else
        {
            printf("\n\n Você está abaixo do peso ideal: %f Kg \n\n", pesoideal);
        }
    }
}
```

// Continua no próximo slide

Solução Exercício 1

24

```
main()
{
    float altura, peso, pesoideal;
    char sexo;
    printf("\n\n Digite sua altura em metros: ");
    altura = lerPesoAltura();
    printf("\n\n Digite seu peso em Kg: ");
    peso = lerPesoAltura();
    sexo = lerSexo();
    pesoideal = calcularPesoldeal(altura, sexo);
    verificarPesoldeal(peso, pesoideal);
    getch();
}
```


Exercício para entregar

25

- Faça um **programa modularizado** que leia a data de nascimento de uma pessoa, ou seja dia, mês e ano, leia o dia, o mês e o ano atual e calcule a idade correta da pessoa. Verifique se ela está fazendo aniversário e mostre na tela a mensagem “Parabéns a você nesta data querida!!!”. O programa deve parar quando for digitada uma data de nascimento inválida.

Modularização de Programa

26

- Os Parâmetros, de uma lista de parâmetros, podem ser recebidos por **Valor** ou por **Referência**.
- **Passagem de Parâmetro por Valor**
 - ▣ Significa que o Módulo recebe o **VALOR** da Informação, ou seja, uma “**cópia**” da informação.
 - ▣ Portanto, se o Módulo alterar essa informação estará **alterando a cópia**.
 - ▣ Sendo assim , a **Informação original**, armazenada na Memória, **estará preservada**.
 - ▣ Passagem de Parâmetro por **VALOR não** permite alterar a **informação original**.

Modularização de Programa

27

□ Passagem de Parâmetro por Referência

- Significa que o Módulo recebe o **ENDEREÇO DE MEMÓRIA** da Informação, ou seja, recebe o acesso direto à informação na memória.
- Portanto, se o Módulo alterar essa informação estará **alterando a informação original**.
- Passagem de Parâmetro por **REFERÊNCIA** permite alterar a **informação original**.
- Para indicar a Passagem de Parâmetros por Referência usa-se o **endereço de memória (&)** ou **ponteiro (*)**.

Modularização de Programa

28

□ Passagem de Parâmetro por Referência

- É importante lembrar que **Variáveis Estruturadas** (Vetor, Matriz e Registro) e **STRING** são sempre passadas por **REFERÊNCIA**, ou seja, sempre que uma Variável Estruturada ou STRING for passada por parâmetro será enviado seu endereço de memória.
- Portanto, qualquer alteração feita pelo Módulo modificará a informação diretamente na memória.
- <https://blog.penjee.com/wp-content/uploads/2015/02/pass-by-reference-vs-pass-by-value-animation.gif>

Modularização de Programa

29

□ Variável Global e Variável Local

- ▣ O uso de modularização de programas nos permite definir, de forma clara, o escopo das variáveis usadas em todo o programa, que pode ser:

- Variável Global
- Variável Local

Modularização de Programa

30

□ Variável Global

- Uma **Variável Global** é declarada, geralmente, após a inclusão das bibliotecas, fora de qualquer Módulo do programa.
- Portanto, é uma variável que pertence a **todos** os Módulos do programa.
- Deste modo, **todos os Módulos podem acessar** (usar e alterar) **seu valor**, pois tem acesso global e seu tempo de permanência na memória é o tempo total de execução do programa.

Modularização de Programa

31

□ **Variável Local**

- ▣ Uma **Variável Local** é uma variável declarada dentro de um bloco de comandos do programa.
- ▣ Este bloco de comandos pode ser:
 - Um Módulo
 - Lista de parâmetros
 - Uma Estrutura de controle de decisão/repetição

Modularização de Programa

32

□ Variável Local – Módulo

```
int somarXY (int X, int Y)
{
    int SOMA; // Variável Local

    SOMA = X + Y;
    return (SOMA);
}
```

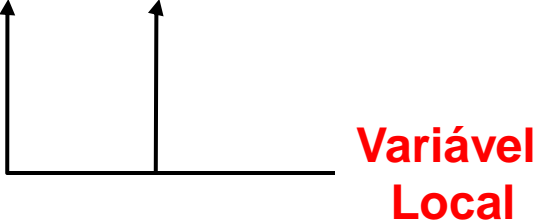

Modularização de Programa

33

□ Variável Local – Lista de Parâmetros

```
int somarXY (int X, int Y)
{
    int SOMA;

    SOMA = X + Y;
    return (SOMA);
}
```



Variável Local

Modularização de Programa

34

□ Variável Local – Uma Estrutura de controle

```
for (int X = 0; X < 100; X = X + 1)
{
    ↑
    └────────────────────────── Variável
                                Local
    printf(" %d ", X) ;
}
```

Modularização de Programa

35

□ Variável Local

- O **acesso** a **Variável é Local**, o que significa que seu **conteúdo só está acessível** dentro do bloco de comandos onde ela foi declarada.
- Seu tempo de **permanência na memória** é o tempo de execução do bloco de comandos onde ela foi declarada.
- Portanto o bloco terminou sua execução, a variável local deixa de existir, seu endereço de memória é liberado para uso.
- Deste modo pode-se ter várias variáveis locais **com mesmo nome**, declaradas em **blocos de comandos diferentes**.
- A variável global é “**única**”, ou seja, não podemos declarar outra variável com seu nome no programa.

Modularização de Programa

36

- Faça um programa modularizado que permite ao usuário escolher uma das opções a seguir:
 - ▣ 1 – Somar dois números inteiros;
 - ▣ 2 – Calcular a tabuada de um número dado;
 - ▣ 3 – Calcular a tabuada do 1 à tabuada do 10;
 - ▣ 4 – Calcular o somatório:

$$\sum_{i=1}^n \frac{i+1}{i^2} = \frac{2}{1} + \frac{3}{4} + \frac{4}{9} + \frac{5}{16} + \dots + \frac{n+1}{n^2}$$

- ▣ 5 – Finalizar o programa.

Exercícios

37

- 2) Faça um **programa modularizado** que carregue um vetor com dez números inteiros e um segundo vetor com cinco números inteiros. Calcule e mostre dois vetores resultantes. O primeiro vetor resultante será composto pelos números pares do primeiro vetor lido. O segundo vetor resultante será composto por todos os números ímpares dos dois vetores lidos.
- 3) Faça um **programa modularizado** que receba o nome e a nota de 80 alunos de uma sala. Calcule e mostre:
- ▣ a média da sala;
 - ▣ o nome do aluno com a maior nota;
 - ▣ o nome do aluno com a menor nota;
 - ▣ os nomes dos alunos aprovados.

Exercícios para Entregar

38

- 1) Em uma eleição presidencial existem quatro candidatos. Os votos são informados por meio de código. Os códigos utilizados são:
 - ▣ 1 – Zé Povim
 - ▣ 2 – João Povão
 - ▣ 3 – Maria do Povo
 - ▣ 4 – Joana da Roça
 - ▣ 5 – Voto nulo
 - ▣ 6 – Voto Branco.
- ▣ Faça um **programa modularizado** que calcule e mostre:
 - ▣ o total de votos para cada candidato;
 - ▣ o total de votos nulos;
 - ▣ o total de votos em branco;
 - ▣ a percentagem de votos nulos sobre o total de votos;
 - ▣ a percentagem de votos em branco sobre o total de votos. Para finalizar o conjunto de votos, tem-se o valor zero.

Bibliografia

39

□ Básica

ASCENCIO, A. F. G, CAMPOS, E. A. V. **Fundamentos da Programação de Computadores**: algoritmos, Pascal e C/C++ e Java. Longman, 2007.

FORBELLONE, L. V., EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. Prentice Hall, 2005.

ZIVIANI, Nivio. **Projeto de Algoritmos com Implementações em Pascal e C**. 2.ed. Thomson Pioneira, 2004.

□ Complementar

FARRER, H et al. **Algoritmos estruturados**. 3 ed. Rio de Janeiro: LTC, 1999. 284 p.

MANZANO, J. A. N. G.; **Estudo dirigido de algoritmos**. 9. ed. São Paulo: Érica, 2004.

LOUNDON, L. **Algoritmos em C**. São Paulo: Ciência Moderna, 2000.

ASCENCIO, A. F. G.; CAMPOS, E.A.V. **Fundamentos da programação de computadores**: algoritmo, pascal e C++. São Paulo: Pearson Prentice Hall, 2002. 355 p