

Modularização – Funções e Procedimentos

Prof. Gabriel Barbosa da Fonseca

Email: gbfonseca@sga.pucminas.br

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

Motivação

- Dividir tarefas (Dividir para conquistar)
 - Programa todo na **main()** fica extenso, confuso e difícil de manter/atualizar
- Reaproveitamento de código

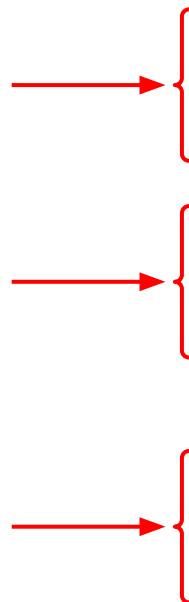
Exemplos

- Programa que lê altura, peso e salário de uma pessoa e garante que os números lidos não são negativos

Motivação

Observam os padrões de repetição??

```
float peso, salario, altura;
do{
    printf("Insira sua altura: \n");
    scanf("%f",&altura);
}while(altura<0);
do{
    printf("Insira seu peso: \n");
    scanf("%f",&peso);
}while(peso<0);
do{
    printf("Insira seu salario: \n");
    scanf("%f",&salario);
}while(salario<0);
```



Exemplos

Funções matemáticas da biblioteca **<math.h>**

- **double pow(double base, double expoente)**
- **double sqrt(double numero)**

Funções/Procedimentos

- Uma função nada mais é do que uma **sub rotina** usada em um programa
- Na linguagem C, denominamos **função** a um conjunto de comandos que realiza uma tarefa específica em um módulo
- A função é referenciada pelo programa principal através do **nome** atribuído a ela

PS: É referenciada pelo nome, similar a uma **variável**

Sintaxe

```
[tipo de retorno] [nome] ([lista de argumentos]) {  
    return [variável de retorno]  
}
```

- [tipo de retorno]: int, char, double, float, ...
- [nome]: O nome da função segue as mesmas regras do nome das variáveis

Sintaxe

```
[tipo de retorno] [nome] ([lista de argumentos]) {  
    return [variável de retorno]  
}
```

- [lista de argumentos]: Pode ter zero ou mais argumentos sendo que cada argumento é composto por seu **tipo** e por uma **variável (nome)**
- Os argumentos são separados por **vírgulas**
- Os argumentos são como **variáveis locais** dentro da função

Sintaxe

```
[tipo de retorno] [nome] ([lista de argumentos]) {  
    return [variável de retorno]  
}
```

- Exemplo: **double** Xablablau(**int** x, **int** y, **double** z, **char** m)
- **Cuidado:** Frequentemente, os alunos erram e colocam (int x, y) <=ERRO!

Sintaxe

```
[tipo de retorno] [nome] ([lista de argumentos]) {  
    return [variável de retorno]  
}
```

Observações:

- O return é facultativo quando o tipo de retorno é **void**
- As linguagens de programação normalmente apresentam duas formas de passagem de parâmetros: **por valor** e **por referência**

Procedimentos

- Quando queremos executar um bloco de comandos, mas estes comandos não precisam retornar nada.
- Neste caso, devemos usar **void** no tipo de retorno do cabeçalho da função.
- Se a função não recebe nenhum parâmetro, também colocamos **void** no local da listagem dos parâmetros.

```
void imprime_cabec(void)
{
    printf("*****\n");
    printf("*      AEDS I – PUC MINAs      *\n");
    printf("*****\n");
}
```

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);
```

```
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Podemos passar uma
constante como
parâmetro?

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);
```

```
    maior = maximo(2345, num1);  
    escrever: "Maior: " + maior;  
    return 0;
```

```
}
```



SIM!

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

Podemos passar um caractere como parâmetro?

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo('A', num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```



SIM!

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);
```

```
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Podemos passar um
double como
parâmetro?

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}  
  
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo(2.655, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Sim! Contudo, a parte decimal será truncada.

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

Podemos retornar
um caractere ou
um double?

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

SIM! Da
mesma forma
que com os
parâmetros

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

Podemos enviar o retorno direto pra tela?

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);  
  
    maior = maximo(num2, num1);  
    escrever: "Maior: " + maior;  
    return 0;  
}
```

Exemplo

```
int maximo(int num1, int num2){  
    int resposta;  
    if (num1 > num2){  
        resposta = num1;  
    } else {  
        resposta = num2;  
    }  
    return resposta;  
}
```

```
int main() {  
    int num1, num2, maior;  
    ler(num1, num2);
```

```
    escrever: "Maior: " + maximo(num2, num1);  
    return 0;
```

```
}
```




SIM

Chamada de funções

- As funções são chamados por outras passando os devidos argumentos

Exemplo: chamando a função Xablablau

```
double Xablablau(int x, int y, double z, char m){  
}
```



```
int a, b;  
double c;  
char d;  
...  
Xablablau(a, b, c, d);  
...
```

Observação 1: O nome das variáveis não precisa ser igual

Observação 2: Não passamos o tipo na chamada (cuidado erro comum)

Exercício

Implemente as seguintes funções. Depois, implemente uma função **main** e faça chamadas às funções implementadas:

```
void imprime_bom_dia(void)
```

```
double log(double numero, double base)
```

```
int soma(float a, float b)
```


Promoção de Argumentos

- **Promoção de Argumentos:** Quando chamamos uma função e passamos para um dos argumentos um valor cujo tipo é mais "fraco" que o esperado, as linguagens C-like simplesmente promovem o tipo desse valor para o esperado
- Por exemplo, `double sqrt(double)`, pode receber um **int** ou um **float**

Truncamento de Argumentos

- **Truncamento de Argumentos:** Quando chamamos uma função e passamos para um dos argumentos um valor cujo tipo é mais “forte” que o esperado:
 - O C e C++ simplesmente truncam o tipo desse valor para o esperado
 - O Java e C# exigem o casting

Truncamento de Argumentos

- As regra de promoção e truncamento também valem para as expressões
- Por exemplo, podemos fazer a seguinte chamada da função:
 - **double sqrt**(double)
 - como: **sqrt**(5 + 5 + 5 + 1)
 - ou: **sqrt**('a' - 32 - 1)

Escopo de Variáveis

- **Variáveis globais:** valem a partir do ponto em que foram declaradas
- **Variáveis locais:** valem dentro do { e } em que foram declaradas

Escopo de Variáveis

- **Variáveis globais:** valem a partir do ponto em que foram declaradas
- **Variáveis locais:** valem dentro do { e } em que foram declaradas

Exercício!

- Faça o quadro de memória e mostre a saída na tela

```
int x = 3;
```

```
void metodo1(){
```

```
    x++;
```

```
}
```

```
void metodo2(int x){
```

```
    x++;
```

```
}
```

```
int main(){
```

```
    escrever: x;
```

```
    x++;
```

```
    escrever: x;
```

```
    metodo1();
```

```
    escrever: x;
```

```
    metodo2(x);
```

```
    escrever: x;
```

```
    return 0;
```

```
}
```

Exercício: Ler x Receber e Mostrar x Retornar

- Faça uma função que leia 2 números e mostre a soma deles
- Faça uma função que leia 2 números e retorne a soma deles
- Faça uma função que receba 2 números e mostre a soma deles
- Faça uma função que receba 2 números e retorne a soma deles

Exercício: Ler x Receber e Mostrar x Retornar

- Faça uma função que **leia** 2 números e **mostre** a soma deles

```
void metodo1(){  
    int n1, n2;  
    ler: n1, n2;  
    escrever: n1+n2;  
}  
  
void main(){  
    metodo1();  
}
```


Exercício: Ler x Receber e Mostrar x Retornar

- Faça uma função que **leia** 2 números e **retorne** a soma deles

```
int metodo2(){  
    int n1, n2;  
    ler: n1, n2;  
    return (n1+n2);  
}  
  
void main(){  
    int resp = metodo2();  
    escrever: resp;  
}
```

Exercício: Ler x Receber e Mostrar x Retornar

- Faça um procedimento que **receba** 2 números e **mostre** a soma deles

```
void metodo3(int n1, int n2){  
    escrever: n1+n2;  
}  
  
void main(){  
    metodo3(5, 3);  
}
```

Exercício: Ler x Receber e Mostrar x Retornar

- Faça uma função que **receba** 2 números e **retorne** a soma deles

```
int metodo4(int n1, int n2){  
    return (n1+n2);  
}  
  
void main(){  
    int n1, n2;  
    ler n1, n2;  
    escrever: metodo4(n1, n2) );  
}
```

Exercício!

- Faça uma função `int multiploCinco(int n)` que recebe um número inteiro `n` e retorna o `n`-ésimo múltiplo de cinco

Exercício!

- Faça uma função `int multiploCinco(int n)` que recebe um número inteiro `n` e retorna o `n`-ésimo múltiplo de cinco

```
int multiploCinco(int n){  
    return n * 5;  
}
```

Exercício!

- Faça uma função ***int multiploTresMaisUm(int n)*** que recebe um número inteiro **n** e retorna o n-ésimo múltiplo de três mais um.

Exercício!

- Faça uma função ***int multiploTresMaisUm(int n)*** que recebe um número inteiro **n** e retorna o n-ésimo múltiplo de três mais um.

```
int multiploTresMaisUm(int n){  
    return (n * 3) + 1;  
}
```

Exercício!

- Faça um procedimento void Exemplo() para ler um número inteiro **n** e mostrar o n-ésimo múltiplo de três mais um que será calculado **usando a função anterior**

Exercício!

- Faça um procedimento void Exemplo() para ler um número inteiro **n** e mostrar o n-ésimo múltiplo de três mais um que será calculado **usando a função anterior**

```
void Exemplo(){  
    int n, multiplo;  
    ler: n;  
    multiplo = multiploTresMaisUm(n);  
    escrever: multiplo;  
}
```