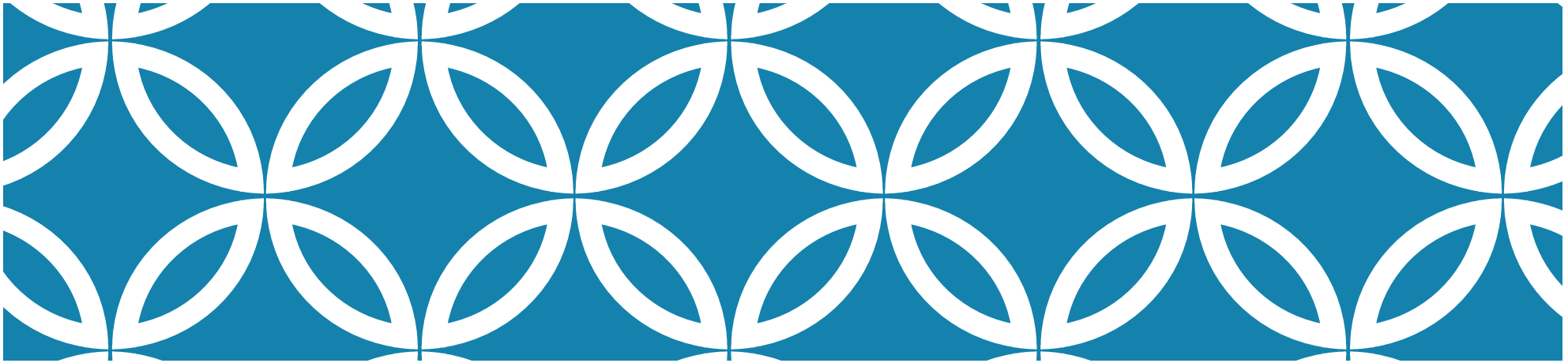

UNIDADE II: FUNÇÕES E PROCEDIMENTOS

Motivação

- Dividir tarefas (Dividir para conquistar)
- Reaproveitamento de código

Algumas Funções Vistos no Curso

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

- **double** sqrt(**double** numero)

Funções

- Uma função nada mais é do que uma subrotina usada em um programa

- Na linguagem C, denominamos função a um conjunto de comandos que realiza uma tarefa específica em um módulo dependente de código

- A função é referenciada pelo programa principal através do nome atribuído a ela

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");
}
```

Nossa 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

Nossa 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
  - Os argumentos são separados por vírgulas
  - Os argumentos são como variáveis locais dentro da função

Nossa Sintaxe

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

- Exemplo: **double** Doidao (**int** x, **int** y, **double** z, **char** m)

- Cuidado: Frequentemente, os alunos erram e colocam (int x, y) <=ERRO!

Nossa 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

Nossa 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

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;
}

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

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

Podemos passar um caractere 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('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;
}

void 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(3.5, 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;
}

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

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

Podemos retornar um caractere ou um double?

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;
}
```

Sim, da mesma forma que na passagem de parâmetros

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 enviar o retorno de uma função diretamente para a tela?

Exemplo

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

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

Sim...

Chamada de Funções

- As funções são chamados por outras passando os devidos argumentos
- Exemplo: chamando a funcaoDoidão

```
...
int a, b;
double c;
char d;
...
funcaoDoidao (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 e teste as funções abaixo:

double log(**double** numero, **double** base)

int arredonda (**float** valor)

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*

Promoção e 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

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

- Faça funções/procedimentos que:
  - Mostre na tela os n primeiros números inteiros, positivos e impares em ordem crescente
  - Mostre na tela os n primeiros números inteiros, positivos e impares em ordem decrescente
  - Retorne o fatorial de n

Exercício

- Faça funções/procedimentos que:
  - Retorne o i-esimo temo da sequência abaixo

$$\frac{1}{3*5}, \frac{2}{3*5^3}, \frac{4}{3*5^9}, \frac{8}{3*5^{27}}, \cdots$$

  - Mostre os n primeiros termos da sequência anterior
  - Efetue o somatório dos n primeiros termos da sequência anterior
  - Efetue o produto dos n primeiros termos da sequência anterior

Exercício

```
int impar(int n){
      int resp = 2 * n + 1;
      return resp;
}

void imparesCrescente int n){
      for(int i = 0; i < n; i++){
            escrever: impar(i);
      }
}

void imparesDecrescente (int n){
      for(int i = n-1; i >= 0; i--){
            escrever: impar(i);
      }
}
```

```
double fatorial(int n){
      double resp = 1;
      for(int i = n; i >= 1; i--){
            resp *= i;
      }
      return resp;
}
double termo(int i){
      double resp = pow(2,i);
      resp  /= (3 * pow(5,(pow(3,i))));
      return resp;
}
void mostrarSequencia(int n){
      for(int i = 0; i <= n; i++){
            escrever: termo(i);
      }
}
```

Exercício

```
double soma(int n){
	double soma = 0;
	for(int i = 0; i <= n; i++){
		soma += termo(i);
	}
	return soma;
}

double produto(int n){
	double produto = 1;
	for(int i = 0; i <= n; i++){
		produto *= termo(i);
	}
	return produto;
}
```

```
void main (){
	int n;
	ler: n;
	imparesCrescente (n);
	imparesDecrescente (n);
	escrever: "fat:" + fatorial(n);
	escrever: "termo:" + termo(n);
	mostrarSequencia(n);
	escrever: "somatorio:" + soma(n);
	escrever: "produto:" + produto(n));
}
```

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) );
}
```

Exemplo

- 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;
}
```

Exemplo

- Faça um procedimento *void exemplo00()* para ler um número inteiro n e mostrar o n-ésimo múltiplo de cinco que será calculado usando a função anterior

```
void exemplo00(){
    int n, multiplo;
    ler(n);
    multiplo = multiploCinco(n);
    escrever: multiplo);
}
```

Exemplo

- Faça um procedimento ***void*** *mostrarMultiploCinco(**int** n)* que recebe um número inteiro **n** e mostra na tela os n primeiros múltiplos de cinco

```
void mostrarMultiploCinco(int n){
    for(int i = 0; i < n; i++){
        escrever: multiploCinco(i));
    }
}
```

Exemplo

- Faça um procedimento ***void*** *exemplo01()* que leia um número inteiro n e chame a função desenvolvida na questão anterior para mostrar os n primeiros múltiplos de cinco

```
void exemplo01(){
    int n;
    ler: n;
    mostrarMultiploCinco(n);
}
```

Exemplo

- 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;
}
```

Exemplo

- Faça um procedimento ***void*** *exemplo02()* para ler um número inteiro n e mostrar o n-ésimo múltiplo de três mais um que será calculado usando a função anterior

```
void exemplo02(){
    int n, multiplo;
    ler: n;
    multiplo = multiploTresMaisUm(n);
    escrever: multiplo;
}
```