

# Procedimentos e Funções

## Algoritmos e Estrutura de Dados I

Instituto de Engenharia – UFMT

# Agenda

- 1 Objetivos
- 2 Introdução
- 3 Por que utilizar funções?
- 4 Declaração de funções
- 5 Declaração de Procedimentos
- 6 Exemplos
- 7 Variáveis Locais x Variáveis Globais

# Objetivos

- Entender e compreender a utilização de funções.
- Aprender a criar suas próprias funções e procedimentos.

# Procedimento

Procedimentos são funções estruturadas que agrupam um conjunto de comandos, que são executados quando o procedimento é chamado. Exemplo

---

```
k = fabs(-1);  
x = sqrt(4);  
y = log10(2);  
z = pow(2,3);
```

---

# Por que utilizar funções?

- Evitar que os blocos do programa fiquem grandes demais (mais difíceis de entender);
- Facilitar a leitura do código-fonte;
- Separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.

# Por que utilizar funções?

- Permitir o reaproveitamento de código já construído (por você ou por outros programadores);
- Evitar que um trecho de código seja repetido várias vezes dentro de um mesmo programa;
- Permitir a alteração de um trecho de código de uma forma mais rápida.

# Declarando uma função



# Declarando uma função

Uma função possui o seguinte formato:

---

```
tipo nome_da_funcao (tipo parametro, tipo parametro, ...) {  
    comandos;  
    return valor;  
}
```

---



# Declarando uma função

- Toda função deve ter um tipo;
- Esse tipo determina qual será o tipo de seu valor de retorno;
- Os parâmetros de uma função determinam qual será o seu comportamento (como se fosse uma função matemática, onde o parâmetro determina o valor da função).

# Declarando uma função

- Uma função pode não ter parâmetros, mas não informá-los;
- A expressão contida dentro do comando `return` é chamado de valor de retorno, e corresponde a resposta de uma determinada função;
- O comando `return` é sempre o último a ser executado por uma função, e nada após ele será executado;
- As funções devem ser declaradas fora do programa principal ( `main()` ).

# Declarando uma função

Exemmplo: a função a seguir soma dois valores passados como parâmetro.

---

```
int soma (int a, int b) {  
    int c = 0;  
    c = a + b;  
    return c;  
}
```

---

que também pode ser feita de forma mais direta:

---

```
int soma (int a, int b) {  
    return a + b;  
}
```

---

# Declarando uma função

Para invocar a função, basta executar o seguinte código:

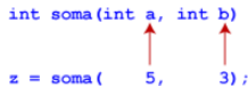
---

```
z = soma (5, 3) ;  
printf("%d", z);
```

---

Que imprimirá 8.

```
int soma(int a, int b)  
  
z = soma( 5, 3);
```

The diagram illustrates the argument passing between a function call and its definition. It shows two lines of code. The first line is the function definition: 'int soma(int a, int b)'. The second line is the function call: 'z = soma( 5, 3);'. Two red arrows point upwards from the arguments '5' and '3' in the function call to the parameters 'a' and 'b' in the function definition, respectively, indicating that the values 5 and 3 are passed to the variables a and b when the function is executed.

# Declarando uma função

- Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

---

```
variavel = funcao (parametros) ;
```

---

- Mas o seu valor também pode ser diretamente impresso:

---

```
printf("%d",funcao(parametros));
```

---

# Declarando uma função

- As variáveis passadas como parâmetros indicam quais são os valores com os quais a função irá trabalhar;
- Esses valores são copiados para os parâmetros da função, que pode manipulá-los;
- Os parâmetros passados para a função não necessariamente possuem os mesmos nomes que os parâmetros que a função espera.

# Declarando uma função

---

```
#include<stdio.h>
int soma (int a, int b) {
    return a + b;
}
int main() {
    int x = 10, y = 30, z;
    z = soma(x,y);
    printf("soma = %d", soma);
    return 0;
}
```

---

# Declarando uma função

- As variáveis declaradas dentro da função só existem na função;
- Os parâmetros passados para a função também só podem ser acessados na função;
- Podem existir variáveis dentro de funções com o mesmo nome de variáveis na função `main()`. Porém tratam-se de posições de memória diferentes.
- Recomenda-se utilizar nomes de variáveis diferentes para evitar confusão.



# Declarando uma função

O seguinte código está ERRADO!

```
#include<stdio.h>
int soma (int a, int b) {
    return a + b;
}
int main() {
    int x = 10, y = 30, z;
    z = soma(x,y);
    printf("a = %d", a);
    printf("b = %d", b);
    return 0;
}
```

- As variáveis a e b são variáveis locais e só podem ser acessadas na função;
- Não é possível acessar variáveis locais dentro do programa principal ( main() ).

## O tipo void

- Tipo especial utilizado principalmente em funções;
- Representa o “nada”;
- Uma variável desse tipo armazena conteúdo indeterminado;
- Uma função desse tipo retorna um conteúdo indeterminado.

# Procedimento em C

- Procedimentos na linguagem C são funções que retornam o tipo void;
- O procedimento abaixo imprime um número que é passado como parâmetro:

---

```
#include<stdio.h>
void imprime (int numero) {
    printf("%d",numero);
}
int main() {
    int x = 10;
    imprime(x);
    return 0;
}
```

---

- Em procedimentos não existe valor de retorno.

## A função main()

- O programa principal ( `main()` ) na verdade é uma função especial;
- A função `main()` é invocada automaticamente pelo programa quando esse inicia sua execução e possui um tipo fixo (`int`);
- O comando `return` informa ao sistema operacional se o programa funcionou corretamente ou não;
- O padrão é que um programa retorne zero caso tenha funcionado corretamente, ou qualquer outro valor caso contrário.

# Declaração de funções/procedimentos

- Normalmente as funções são definidas no início do código, antes da função main();
- Dessa forma, a função é declarada e também implementada;
- Também é possível definir uma função no fim do código, após a função main();
- No entanto, a função deve ser declarada no início do código, e sua implementação fica no fim do código.
- Para declará-la sem a implementação, basta incluir a primeira linha da função, com um ponto e vírgula no final.

---

```
int nome_da_funcao(tipo parametro, tipo parametro, ... );
```

---

# Declaração de funções/procedimentos

```
#include<stdio.h>
int soma (int a, int b);
int main() {
    int x = 10, y = 30, z;
    z = soma(x,y);
    printf("%d",z);
    return 0;
}
int soma (int a, int b) {
    return a + b;
}
```

# Declaração de funções/procedimentos

Crie uma função que retorna 1 se um número é par, e 0 caso contrário.

# Declaração de funções/procedimentos

---

```
#include<stdio.h>
int par (int x);
int main() {
    int numero;
    scan("%d", numero);
    if ( par(numero) == 1)
        printf("Eh par.");
    else
        printf("Eh impar.");
    return 0;
}
int par (int x) {
    if (x%2==0)
        return 1; // Verdadeiro
    else
        return 0; // Falso
}
```



# Declaração de funções/procedimentos

Leia dois catetos e crie uma função para calcular a hipotenusa e um procedimento para imprimir o seu valor.

# Declaração de funções/procedimentos

```
#include<stdio.h>
float pitagoras (float cat1, float cat2);
void imprime(float x);
int main() {
    float cat1, cat2, hipo;
    printf("Digite os valores dos catetos:");
    scanf("%f %f",&cat1, &cat2);
    hipo = pitagoras(cat1, cat2);
    imprime(hipo)
    return 0;
}
```

# Declaração de funções/procedimentos

---

```
float pitagoras (float cat1, float cat2) {  
    return sqrt( pow(cat1,2) + pow(cat2,2));  
}  
  
void imprime(float valor) {  
    printf("Hipotenusa = %f", valor);  
}
```

---

# Declaração de funções/procedimentos

Faça uma função que retorne 1 caso um número seja primo, ou 0 caso contrário.

# Declaração de funções/procedimentos

```
#include<stdio.h>
int eh_primo(int x);
int main() {
    int n
    scanf("%d",&n);
    if (eh_primo(n) == 1)
        printf("Eh primo\n");
    else
        printf("Nao eh primo\n");
    return 0;
}
```

# Declaração de funções/procedimentos

```
#include<stdio.h>
int eh_primo(int numero) {
    int cont = 0, i;
    for (i=2,i<numero;i++) {
        if (numero%i==0)
            cont++;
    }
    if (cont == 0)
        return 1; // Verdadeiro
    else
        return 0; // Falso
}
```

# Variáveis Locais

- Variáveis Locais

- ▶ É declarada dentro de uma função;
- ▶ Existe apenas dentro da função;
- ▶ Após o término da execução da função, ela deixa de existir.

# Variáveis Globais

- Variáveis Globais

- ▶ É declarada fora de qualquer função, incluindo a função `main()`;
- ▶ Existe dentro de qualquer função e procedimento;
- ▶ Qualquer função ou procedimento pode alterar seu valor.
- ▶ Evite utilizar variáveis globais;



# Variáveis Locais x Variáveis Globais

```
#include<stdio.h>
int var_global = 0;
void incrementa_var_global(int i) {
    var_global+=i;
}
int main() {
    printf("Var global = %d", var_global);
    incrementa_var_global(10);
    printf("Var global = %d", var_global);
    return 0;
}
```

Imprime:

Var global = 0

Var global = 10

Fim

Fim