

LPG0001 – Linguagem de Programação

Funções

Prof. Rui Jorge Tramontin Junior
Departamento de Ciência da Computação
UDESC / Joinville

Introdução

- Funções são blocos de código reutilizáveis;

Introdução

- Funções são blocos de código reutilizáveis;
- Podem receber parâmetros → entrada;

Introdução

- Funções são blocos de código reutilizáveis;
- Podem receber parâmetros → entrada;
- Podem retornar um valor → saída;

Introdução

- Podem ter suas próprias variáveis (locais), o que facilita a organização do código;

Introdução

- Podem ter suas próprias variáveis (locais), o que facilita a organização do código;
- Auxiliam no entendimento de um problema, que pode ser decomposto em partes menores que são tratadas por funções específicas.

Sintaxe da declaração de funções

```
tipo_retorno  nome_da_função  ( parâmetros ) {  
  
    /*  
        Declarações de variáveis  
        e comandos...  
    */  
  
}
```

Sintaxe da declaração de funções

```
tipo_retorno nome_da_função ( parâmetros ) {
```

```
/*
```

```
Declarações de variáveis  
e comandos...
```

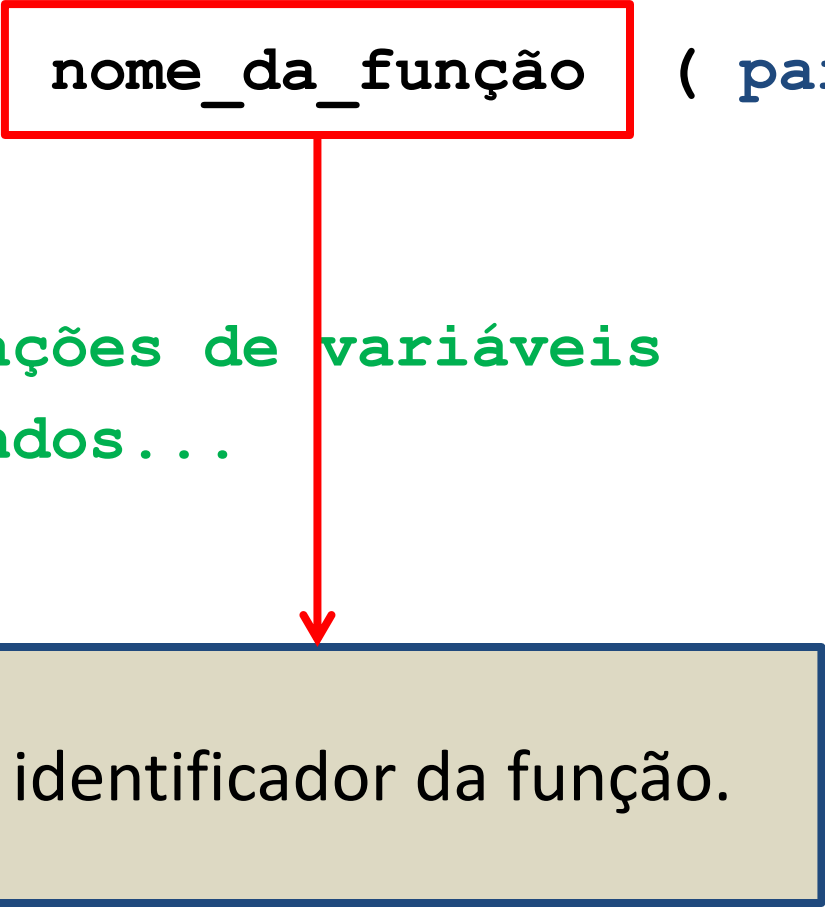
```
*/
```

```
}
```

É um tipo da linguagem (*int*, *float*, *void*, etc.) e define o tipo de valor que a função retorna.

Sintaxe da declaração de funções

```
tipo_retorno nome_da_função ( parâmetros ) {  
  
    /*  
        Declarações de variáveis  
        e comandos...  
    */  
  
}
```



É o identificador da função.

Sintaxe da declaração de funções

```
tipo_retorno  nome_da_função  ( parâmetros ) {  
  
    /*  
        Declarações de variáveis  
        e comandos...  
    */  
  
}
```

Lista de declarações semelhantes a declarações de variáveis. A lista pode estar vazia (neste caso, a função não tem parâmetros).

Sintaxe da declaração de funções

```
tipo_retorno  nome_da_função  ( parâmetros ) {
```

```
/*
```

```
    Declarações de variáveis  
    e comandos...
```

```
*/
```

```
}
```

Código da função; contém suas próprias variáveis e pode conter um ou mais comandos *return*, para retornar o valor calculado.

EXEMPLO 1: QUADRADO DE UM NÚMERO

Exemplo 1: quadrado de um número

```
float quadrado( float x ){  
    return x * x;  
}
```

Exemplo 1: quadrado de um número

```
float quadrado( float x ){  
    return x * x;  
}
```

```
int main(){  
  
    return 0;  
}
```

Exemplo 1: quadrado de um número

```
float quadrado( float x ){  
    return x * x;  
}
```

```
int main(){  
    float a;  
    scanf("%f", &a);  
  
    return 0;  
}
```

Exemplo 1: quadrado de um número

```
float quadrado( float x ){  
    return x * x;  
}
```

```
int main(){  
    float a;  
    scanf("%f", &a);  
    float b = quadrado( a ); // chamada da função  
  
    return 0;  
}
```


Exemplo 1: quadrado de um número

```
float quadrado( float x ){  
    return x * x;  
}
```

```
int main(){  
    float a;  
    scanf("%f", &a);  
    float b = quadrado( a ); // chamada da função  
    printf("%f ao quadrado = %f\n", a, b );  
  
    return 0;  
}
```

Considerações

- Tal como variáveis, funções devem ser declaradas previamente ao ponto do código onde ocorre sua chamada;

Considerações

- Tal como variáveis, funções devem ser declaradas previamente ao ponto do código onde ocorre sua chamada;
- Quando uma função é chamada, o fluxo de execução é desviado para a sua implementação;

Considerações

- Tal como variáveis, funções devem ser declaradas previamente ao ponto do código onde ocorre sua chamada;
- Quando uma função é chamada, o fluxo de execução é desviado para a sua implementação;
- O valor passado para a função (**variável a**) é carregado no parâmetro (**x**);

Considerações

- A função executa até encontrar o comando *return*;

Considerações

- A função executa até encontrar o comando *return*;
- A execução retorna para onde a função foi chamada;

Considerações

- A função executa até encontrar o comando *return*;
- A execução retorna para onde a função foi chamada;
- O valor retornado neste exemplo foi atribuído à *variável b*;

Considerações

- Um aspecto interessante a respeito de funções é que é possível imprimir diretamente o resultado da função na tela;

Considerações

- Um aspecto interessante a respeito de funções é que é possível imprimir diretamente o resultado da função na tela;
- Neste caso, ao invés de usarmos a variável b , poder-se-ia fazer da seguinte forma:

Considerações

- Um aspecto interessante a respeito de funções é que é possível imprimir diretamente o resultado da função na tela;
- Neste caso, ao invés de usarmos a variável b , poder-se-ia fazer da seguinte forma:

```
printf("%f ao quadrado = %f\n", a, quadrado(a));
```

EXEMPLO 2: MAIOR ENTRE DOIS NÚMEROS

Exemplo 2: maior entre dois números

```
int maior( int x, int y ){  
    if( x > y)  
        return x;  
    else  
        return y;  
}
```

Exemplo 2: maior entre dois números

```
int maior( int x, int y ){  
    if( x > y)  
        return x;  
    else  
        return y;  
}
```

```
int main(){
```

```
}
```

Exemplo 2: maior entre dois números

```
int maior( int x, int y ){  
    if( x > y)  
        return x;  
    else  
        return y;  
}
```

```
int main(){  
    int a, b;  
    scanf("%i%i", &a, &b);  
  
}
```

Exemplo 2: maior entre dois números

```
int maior( int x, int y ){  
    if( x > y)  
        return x;  
    else  
        return y;  
}
```

```
int main(){  
    int a, b;  
    scanf("%i%i", &a, &b);  
    printf("Maior valor =\n", maior( a, b ) );  
    return 0;  
}
```

Considerações

- Este exemplo ilustra uma função que possui mais de um comando *return*;

Considerações

- Este exemplo ilustra uma função que possui mais de um comando *return*;
- Além disso, aqui a função foi chamada dentro do *printf*;
 - Seu retorno foi mostrado diretamente no console;

Considerações

- Este exemplo ilustra uma função que possui mais de um comando *return*;
- Além disso, aqui a função foi chamada dentro do *printf*;
 - Seu retorno foi mostrado diretamente no console;
- Funções podem ser declaradas separadamente de sua implementação, conforme mostrado no próximo exemplo.

EXEMPLO 3: CÁLCULO DA POTENCIAÇÃO

Exemplo 3: cálculo da potenciação

```
float potencia ( float base, int expo ); // declaração
```

```
int main() {
```

```
    float b;
```

```
    int e;
```

```
    printf("Digite dois números: ");
```

```
    scanf("%f%d", &b, &e);
```

```
    float p = potencia( b, e ); // chamada da função
```

```
    printf("%f elevado a %d = %f\n", b, e, p);
```

```
    return 0;
```

```
}
```

Exemplo 3: cálculo da potenciação

```
float potencia ( float base, int expo ){  
    // inverte-se a base, caso expo seja negativo  
    if( expo < 0 ){  
        base = 1 / base; //ex:  $2^{-3} = 1/2 \cdot 1/2 \cdot 1/2 = 1/8$   
        expo = -expo;  
    }  
  
    // cálculo da potenciação  
    float pot = 1;  
    while( expo > 0 ){  
        pot = pot * base;  
        expo--;  
    }  
    return pot;  
}
```

Considerações sobre o exemplo

- A função *potencia* recebe dois parâmetros (*base* e *expo*), cujos valores são oriundos das variáveis *b* e *e*, espectivamente;

Considerações sobre o exemplo

- A função *potencia* recebe dois parâmetros (*base* e *expo*), cujos valores são oriundos das variáveis *b* e *e*, espectivamente;
- O retorno da função é atribuído à variável *p*;

Considerações sobre o exemplo

- A função *potencia* recebe dois parâmetros (*base* e *expo*), cujos valores são oriundos das variáveis *b* e *e*, respectivamente;
- O retorno da função é atribuído à variável *p*;
- O primeiro *if* inverte a base caso o expoente seja negativo;

Considerações sobre o exemplo

- A função *potencia* recebe dois parâmetros (*base* e *expo*), cujos valores são oriundos das variáveis *b* e *e*, respectivamente;
- O retorno da função é atribuído à variável *p*;
- O primeiro *if* inverte a base caso o expoente seja negativo;
- Por exemplo: $2^{-3} = 1/2 \cdot 1/2 \cdot 1/2 = 1/8$.

Considerações sobre o exemplo

- Perceba que a função sempre vai zerar o valor do parâmetro *expo*.

Considerações sobre o exemplo

- Perceba que a função sempre vai zerar o valor do parâmetro *expo*.
- Porém, a variável *e*, da função *main*, permanece inalterada, como pode ser verificado quando o programa mostra o resultado.

Considerações sobre o exemplo

- Perceba que a função sempre vai zerar o valor do parâmetro *expo*.
- Porém, a variável *e*, da função *main*, permanece inalterada, como pode ser verificado quando o programa mostra o resultado.
 - Passagem de parâmetro por valor;

Considerações sobre o exemplo

- Perceba que a função sempre vai zerar o valor do parâmetro *expo*.
- Porém, a variável *e*, da função *main*, permanece inalterada, como pode ser verificado quando o programa mostra o resultado.
 - Passagem de parâmetro por valor;
- Como somente o valor da variável está sendo copiado, é possível passar literalmente um valor para a função.

Considerações sobre o exemplo

- Por exemplo, se quiséssemos calcular 2^4 diretamente no programa, seria possível escrever da seguinte forma:

Considerações sobre o exemplo

- Por exemplo, se quiséssemos calcular 2^4 diretamente no programa, seria possível escrever da seguinte forma:

```
float p = potencia( 2, 4 );
```

FUNÇÕES BOOLEANAS

Lembrete sobre o tipo booleano em C

- É importante destacar que a linguagem C não tem um tipo específico para valores lógicos (booleanos);

Lembrete sobre o tipo booleano em C

- É importante destacar que a linguagem C não tem um tipo específico para valores lógicos (booleanos);
- Usa-se o tipo *int*, sendo o valor 0 interpretado como falso e 1 como verdadeiro.

Exemplo 4: verifica se *char* é uma letra

```
int eh_letra ( char x ); // declaração da função
```

```
int main() {  
    char k;  
    printf("Digite um caractere: ");  
    scanf("%c", &k);  
    if( eh_letra( k ) ) // chamada da função  
        printf("%c é letra!\n", k);  
    else  
        printf("%c não é letra!\n", k);  
    return 0;  
}
```

Exemplo 4: verifica se *char* é uma letra

// implementação da função

```
int eh_letra ( char x ) {  
    if( x >= 'a' && x <= 'z' )  
        return 1;  
    else  
        return 0;  
}
```

Considerações

- Na chamada da função (`if(eh_letra(k))`), o fluxo de execução é desviado para a implementação da mesma

Considerações

- Na chamada da função (`if(eh_letra(k))`), o fluxo de execução é desviado para a implementação da mesma
- O valor passado para a função (`variável k`) é atribuído ao parâmetro `x` da função.

Considerações

- Na chamada da função (`if(eh_letra(k))`), o fluxo de execução é desviado para a implementação da mesma
- O valor passado para a função (`variável k`) é atribuído ao parâmetro `x` da função.
- De acordo com o valor passado como parâmetro, a função retorna 0 ou 1, que é então tratado pela estrutura *if*.

Considerações

- Uma forma mais enxuta para implementar funções booleanas consiste em retornar diretamente a expressão lógica;

Considerações

- Uma forma mais enxuta para implementar funções booleanas consiste em retornar diretamente a expressão lógica;

```
int eh_letra ( char x ) {  
    return x >= 'a' && x <= 'z' ;  
}
```

Considerações

- Uma forma mais enxuta para implementar funções booleanas consiste em retornar diretamente a expressão lógica;

```
int eh_letra ( char x ) {  
    return x >= 'a' && x <= 'z' ;  
}
```

- A expressão lógica será avaliada como verdadeiro (1) ou falso (0), valor que será retornado diretamente pela função.

EXERCÍCIO EM AULA

Exercício em aula

- Faça um programa que lê os três lados de um triângulo e determina o seu tipo (equilátero, isóceles ou escaleno).
- Utilize uma função que recebe os três lados e retorna o tipo de triângulo. A função poderia ser, por exemplo:

```
int tipo_triangulo(float x, float y, float z);  
// 1 - equilátero, 2 - isóceles ou 3 - escaleno
```