



Linguagem C

Funções

MsC. Douglas Santiago Kridi

Programação I - 2018.2

Bacharelado em Ciência da Computação

Universidade Estadual do Piauí

douglaskridi@gmail.com

Introdução

- Funções (também chamadas de *subrotinas*).
 - São trechos de código situados fora do programa principal.
 - As funções são identificadas por um nome.
 - Este nome é chamado (ou invocado) toda vez que se desejar executar o trecho de código da função.

Introdução

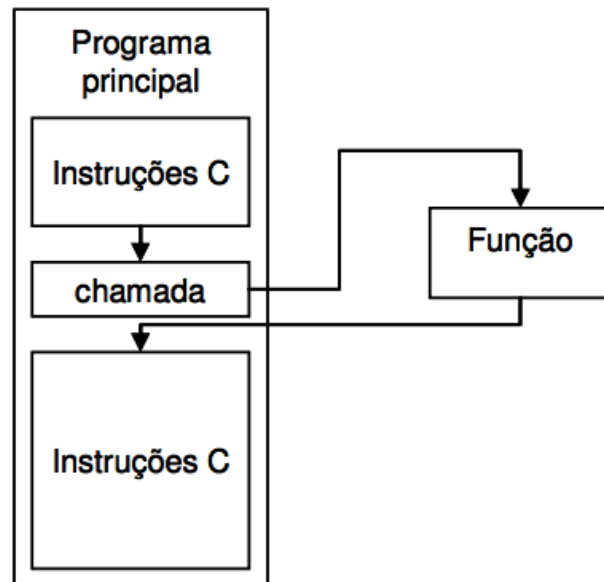
- Uma função implementa um algoritmo, e, portanto, apresenta os mesmos conceitos do algoritmo:
 - entrada, saída, código finito.
- Uma função é um bloco de código que:
 - Realiza uma determinada *tarefa específica*.
 - Pode ser *executado a qualquer momento* pelo programa.
 - Recebe dados de entrada do programa (ou do teclado, arquivo, etc).
 - Retorna um resultado (ou escreve algo na tela, em arquivo, etc).

Introdução

- Um programa pode executar uma função diversas vezes, em momentos diferentes, com valores diferentes para as variáveis.
 - Ideal quando temos uma mesma tarefa que precisamos executar em diversos pontos diferentes do programa.
 - Basta *chamar a função*, ao invés de repetir todo seu código.
 - Os valores que a função deve utilizar naquele momento são dados de entrada que são passados como *parâmetros*.

Introdução

■ Funcionamento de uma função:



■ Quando a execução chega a uma instrução contendo uma chamada de função, **a execução do programa que chamou é interrompida** temporariamente e o computador passa a executar as instruções contidas na função.


■ Terminada a execução da função, o código principal **continua na mesma posição em que o programa foi interrompido**, e ainda disponibiliza um **valor de retorno** que pode ser atribuído a uma variável de mesmo tipo.

Exemplo

- **Problema:** para dois alunos, o programa lê o valor de três notas. A média é calculada somente com as duas melhores. Deseja-se descobrir qual dos dois alunos obteve a melhor média.

```
// Declarar notas e médias dos alunos A e B
float notaA1, notaA2, notaA3, mediaA;
float notaB1, notaB2, notaB3, mediaB;
// Ler notas dos alunos A e B
scanf("%f %f %f", &notaA1, &notaA2, &notaA3);
scanf("%f %f %f", &notaB1, &notaB2, &notaB3);
// Calcular a média para o aluno A
if ((notaA1 <= notaA2) && (notaA1 <= notaA3)) {
    mediaA = (notaA2 + notaA3) / 2.0;
} else if ((notaA2 <= notaA1) && (notaA2 <= notaA3)) {
    mediaA = (notaA1 + notaA3) / 2.0;
} else {
    mediaA = (notaA1 + notaA2) / 2.0; }

// Calcular a média para o aluno B
if ((notaB1 <= notaB2) && (notaB1 <= notaB3)) {
    mediaB = (notaB2 + notaB3) / 2.0;
} else if ((notaB2 <= notaB1) && (notaB2 <= notaB3)) {
    mediaB = (notaB1 + notaB3) / 2.0;
} else {
    mediaB = (notaB1 + notaB2) / 2.0; }
// Imprimir resultado
if (mediaA > mediaB) {
    printf("Aluno A obteve melhor desempenho");
} else if (mediaA < mediaB) {
    printf("Aluno B obteve melhor desempenho");
} else {
    printf("Os alunos A e B tem a mesma media"); }
```

A dashed orange box encloses the code for calculating the average for student A and the logic for comparing the averages of students A and B. An orange arrow points from the end of the first code block (the calculation of mediaA) to the start of the second code block (the comparison logic).

Exemplo

- Alguns problemas na abordagem anterior:
 - *Repete as mesmas instruções* para calcular as médias de A e B.
 - *Repete o mesmo algoritmo*, mas para variáveis diferentes.
 - Se for necessário *alterar uma instrução*, é preciso garantir que as duas cópias do código sejam mudadas igualmente, para se ter *consistência*.
 - Facilmente alguém esquecerá de alterar algum detalhe em uma das repetições do código.
 - Por fim, é difícil *reaproveitar* o algoritmo.

Corpo de uma função

- Uma função é definida da seguinte forma:

```
tipo nome(parâmetros) {  
    // declaração de variáveis  
  
    // corpo da função  
    return valor;  
}
```

- Exemplo: Média dos alunos

```
Tipo mediaMelhoresNotas(...) {  
    // Corpo da função  
}
```

Nome da função

Códigos que serão executados
pela função

Corpo de uma função

- A maioria das funções recebe pelo menos um valor como entrada.
 - Estes valores são os **parâmetros**.
- Na definição da função, um parâmetro *é uma variável especial, que existe somente dentro da função*.
 - Os parâmetros (e seus respectivos tipos) são informados em uma lista *separada por vírgulas e entre parênteses*, logo após o nome da função.

```
Tipo mediaMelhoresNotas(float nota1, float nota2, float nota3) {  
    // Corpo da função  
}
```

Lista de parâmetros

Corpo de uma função

- Quando a função é chamada, deve-se informar os valores de cada um dos parâmetros, na mesma ordem que aparecem na definição.
 - Os valores precisam ter tipo compatível com o tipo do respectivo parâmetro.

- Exemplo:

```
Tipo potencia(float base, float expoente) {  
    // Corpo da função  
}
```

- Poderíamos chamar como:

```
float resultado;  
resultado = potencia(2.0, 5.0);
```

Corpo de uma função

- Existem casos especiais de funções que não têm parâmetros.
 - Nestas situações, a lista de parâmetros é vazia.
 - Mas, mesmo assim, deve-se escrever os parênteses, tanto na definição, quanto na chamada da função.

```
Tipo funcaoSemParametros() {  
    // Corpo da função  
}
```

Corpo de uma função

- No início do corpo da função, é comum declarar *variáveis locais*.
 - Elas *são conhecidas somente dentro do corpo da função*.
 - Em qualquer outro ponto do programa, estas variáveis não são conhecidas e será impossível acessar o valor das mesmas.

```
Tipo mediaMelhoresNotas( float nota1, float nota2, float nota3) {  
    float media;  
    if ((nota1 <= nota2) && (nota1 <= nota3)) {  
        media = (nota2 + nota3) / 2.0;  
    } else if ((nota2 <= nota1) && (nota2 <= nota3)) {  
        media = (nota1 + nota3) / 2.0;  
    } else {  
        media = (nota1 + nota2) / 2.0; }  
    // retorno da função  
}
```

Variáveis locais e globais

- A *variável global* é aquela declarada fora de qualquer função, tipicamente no início do arquivo do código fonte.
- A *variável local* é declarada dentro do corpo de uma função, ou como parâmetro.

Variáveis locais e globais

■ Variável Global

```
#include <stdlib.h>
#include <stdio.h>

int v = 0;

void f1() {
    v++;
    printf("f1: v = %d\n", v);
}

void f2() {
    v+=2;
    printf("f2: v = %d\n", v);
}

int main() {
    f1(); f2(); f1();
    return 0;
}
```

O resultado será:

f1: v = 1

f2: v = 3

f1: v = 4

Variáveis locais e globais

■ Variável Local

```
#include <stdlib.h>
#include <stdio.h>

void f1() {
    int v = 0;
    v++;
    printf("f1: v = %d\n", v);
}

void f2() {
    int v = 0;
    v+=2;
    printf("f2: v = %d\n", v);
}

int main() {
    f1(); f2(); f1();
    return 0;
}
```

O resultado será:

f1: v = 1

f2: v = 2

f1: v = 1

Corpo de uma função

- As funções produzem um resultado.
 - O comando *return termina a execução da função* e define o valor de retorno.
 - Na chamada da função, o código chamador pode atribuir este valor retornado a uma variável ou usá-lo em uma expressão.

ex.: `int var = funcao();`

- Na especificação da função, precisamos indicar o tipo de retorno.

```
float mediaMelhoresNotas( float nota1, float nota2, float nota3) {  
    float media;  
    if ((nota1 <= nota2) && (nota1 <= nota3)) {  
        media = (nota2 + nota3) / 2.0;  
    } else if ((nota2 <= nota1) && (nota2 <= nota3)) {  
        media = (nota1 + nota3) / 2.0;  
    } else {  
        media = (nota1 + nota2) / 2.0; }  
    return media;  
}
```


Corpo de uma função

- O comando ***return*** não necessariamente precisa ser a última instrução da função.
 - Observe no exemplo abaixo que apenas um return será executado:

```
float mediaMelhoresNotas( float nota1, float nota2, float nota3) {  
    float media;  
    if ((nota1 <= nota2) && (nota1 <= nota3)) {  
        return (nota2 + nota3) / 2.0;  
    } else if ((nota2 <= nota1) && (nota2 <= nota3)) {  
        return (nota1 + nota3) / 2.0;  
    } else {  
        return (nota1 + nota2) / 2.0; }  
}
```

Corpo de uma função

- Em casos especiais, a função apenas executa o código, mas não gera resultado que possa ser retornado.
 - Nesses casos, o tipo da função será **void**.
 - O comando **return** *passa a ser opcional* e pode ser chamado sem valor de retorno, apenas para terminar a função

```
void comparaNotas(float mediaA, float mediaB) {  
    if (mediaA > mediaB) {  
        printf("Aluno A obteve melhor desempenho");  
    } else if (mediaA < mediaB) {  
        printf("Aluno B obteve melhor desempenho");  
    } else {  
        printf("Os alunos A e B tem a mesma media");  
    }  
}
```

- A princípio, funções sem retorno seriam procedimentos, mas em C, não há essa distinção.

Exemplo: Media dos alunos

- Função para o calculo da média:

```
float mediaMelhoresNotas(float nota1, float nota2, float nota3) {  
    float media;  
  
    if ((nota1 <= nota2) && (nota1 <= nota3)) {  
        media = (nota2 + nota3) / 2.0;  
    } else if ((nota2 <= nota1) && (nota2 <= nota3)) {  
        media = (nota1 + nota3) / 2.0;  
    } else {  
        media = (nota1 + nota2) / 2.0;  
    }  
    return media;  
}
```

Exemplo: Media dos alunos

- Função para determinar aluno com maior média:

```
void comparaNotas(float mediaA, float mediaB) {  
    if (mediaA > mediaB) {  
        printf("Aluno A obteve melhor desempenho");  
    } else if (mediaA < mediaB) {  
        printf("Aluno B obteve melhor desempenho");  
    } else {  
        printf("Os alunos A e B tem a mesma media");  
    }  
}
```

Exemplo: Media dos alunos

■ Programa principal (função Main):


```
int main() {  
    // Declarar notas e médias dos alunos A e B  
    float notaA1, notaA2, notaA3, mediaA;  
    float notaB1, notaB2, notaB3, mediaB;  
  
    // Ler notas dos alunos A e B  
    scanf("%f %f %f", &notaA1, &notaA2, &notaA3);  
    scanf("%f %f %f", &notaB1, &notaB2, &notaB3);  
  
    // Calcular a média para o aluno A  
    mediaA = mediaMelhoresNotas(notaA1, notaA2, notaA3);  
    // Calcular a média para o aluno B  
    mediaB = mediaMelhoresNotas(notaB1, notaB2, notaB3);  
  
    // Imprimir resultado  
    comparaNotas(mediaA, mediaB);  
    return 0;  
}
```

Exemplo: Números pares

- Fazer um programa para verificar se um dado número é par ou não. Para isso, utilize uma função com passagens de parâmetros.

```
int verifPar(int k) {  
    if (k%2 == 0)  
        return 1;  
    else  
        return 0;  
}
```

```
int main() {  
    int num, resultado;  
    printf("Digite um numero ");  
    scanf("%d",&num);  
  
    resultado = verifPar(num);  
  
    if(resultado == 1)  
        printf("\nO numero que voce digitou é par\n");  
    else  
        printf("\nO numero que voce digitou é ímpar\n");  
  
    return 0;  
}
```

A dashed orange box encloses both code snippets. An orange arrow originates from the `verifPar(num)` call in the `main` function and points to the `verifPar` function definition, illustrating the function call mechanism.

Posição da função

- Quando um programa usa funções, escrevemos o código das funções *antes do código* da rotina principal.
- No entanto, o programa ficaria mais bem organizado se pudéssemos escrever o código da rotina principal antes do código da função auxiliar.
 - A solução é declarar antes do main() apenas o *protótipo* de cada função, e a função em si é declarada após a main().
 - Para declarar um protótipo, basta pegarmos o cabeçalho de cada função e adicionar um ponto-e-vírgula ao fim deste.

Posição da função


■ Exemplo

```
#include <stdio.h>

int teste(int a);

int main(void) {
    int a = 1;
    printf("Valor inicial de 'a': %d\n", a);
    a=teste(a);
    printf("Valor de 'a' apos receber o return da funcao: %d\n", a);
}

int teste(int a) {
    a++;
    printf("Estou dentro da funcao!");
    return a;
}
```

A thick orange bracket on the left side of the code block connects the call to the function `a=teste(a);` inside the `main` function to the definition of the `teste` function below it, illustrating the flow of control.

Pratique:

1. Crie uma função em linguagem C que receba 2 números e retorne o maior valor.
2. Crie uma função que receba um valor e informe se ele é positivo ou não.
3. Crie um aplicativo de conversão entre as temperaturas Celsius e Farenheit.
 - a) Primeiro o usuário deve escolher se vai entrar com a temperatura em Célsius ou Farenheit, depois a conversão escolhida é realizada através de um comando SWITCH.
 - b) Se C é a temperatura em Célsius e F em farenheit, as fórmulas de conversão são:
 $C = 5.(F-32)/9$, $F = (9.C/5) + 32$
4. Um professor, muito legal, fez 3 provas durante um semestre mas só vai levar em conta as duas notas mais altas para calcular a média.
 - a) Faça uma aplicação em C que peça o valor das 3 notas, mostre como seria a média com essas 3 provas, a média com as 2 notas mais altas, bem como sua nota mais alta e sua nota mais baixa.

Bibliografia Básica



C: COMO PROGRAMAR

DEITEL, Harvey M.; DEITEL, Paul J. Editora Pearson - 6ª ed. 2011

Fundamentos da Programação de Computadores

Ascencio, Ana F. G., Campos, Edilene A. V. de, - Editora Pearson

2012



Lógica de Programação e Estrutura de Dados

Puga, Sandra. Riseti, Gerson. – Ed. Pearson - 2016

Lógica de Programação Algorítmica (Apostila)

Guedes, Sergio. - Editora Pearson/Ser - 2014

