Fundamentos de Programação

Aula 07 - Funções

Arnaldo Barreto Vila Nova

Funções

- Conceito
- Tipos
- Parâmetros - Passagem por valor
- Parâmetros - Passagem por referência
- Escopo de Variáveis

Funções - Conceito

- Uma função é um bloco de comandos que recebe um nome e pode ser ativado através deste nome
- Sintaxe

```
tipo nome(parâmetros)
{
    \\bloco de comandos
    return valor;
}
```

Funções - Motivos

- Alguns motivos para utilizar funções:
  - Evitar trechos repetidos várias vezes no programa
  - Facilitar a alteração de determinado trecho do código
  - Facilitar a leitura do código
  - Separar o programa em partes independentes

Funções - Motivos

- Exemplo:
    - Ler 4 números (a, b, c e d), calcular a média dos dois primeiros, calcular a média dos dois últimos, e mostrar média das médias.
        - Várias médias calculadas → fazer função para calcular média

Funções - Motivos

- Exemplo:
    - Ler 4 números (a, b, c e d), calcular a média dos dois primeiros, calcular a média dos dois últimos, e mostrar média das médias.
        - Várias médias calculadas → fazer função para calcular média

```
float calcular_media(float x, float y)
{
    float media = (x+y)/2.0;
    return media;
}
```

Funções - Tipos

- O tipo de uma função indica o tipo do valor que a função vai retornar
  - **int** → retorna inteiro; **char** → retorna caracter; ...
  - **void** → não retorna nenhum valor

```
float somar(float x, float y)
{
    return x + y;
}
```

```
void somar(float x, float y)
{
    printf("x+y = %f",x+y);
}
```

Funções - Tipos

- O tipo de uma função indica o tipo do valor que a função vai retornar
  - **int** → retorna inteiro; **char** → retorna caracter; ...
  - **void** → não retorna nenhum valor

```
float somar(float x, float y)
{
    return x + y;
}
```

Função float precisa de um retorno do tipo float

```
void somar(float x, float y)
{
    printf("x+y = %f",x+y);
}
```

Função void não tem retorno

Funções - Retorno

- **return** é o comando que define o valor resultante da função
  - O valor retornado deve ser do tipo da função
  - Só é possível retornar 1 único valor
  - O return finaliza a função (nada após ele vai ser executado)

```
float somar(float x, float y)
{
    return x + y;
}
```

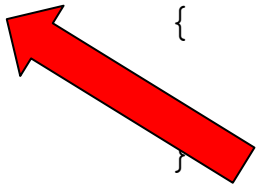
```
void main()
{
    float x = somar(4.5, 7.1);
}
```

Funções - Retorno

- **return** é o comando que define o valor resultante da função
  - O valor retornado deve ser do tipo da função
  - Só é possível retornar 1 único valor
  - O return finaliza a função (nada após ele vai ser executado)

```
float somar(float x, float y)
{
    return x + y;
}
```

```
void main()
{
    float x = somar(4.5, 7.1);
}
```



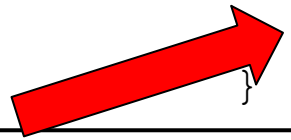
A chamada passa os valores para a função

Funções - Retorno

- **return** é o comando que define o valor resultante da função
  - O valor retornado deve ser do tipo da função
  - Só é possível retornar 1 único valor
  - O return finaliza a função (nada após ele vai ser executado)

```
float somar(float x, float y)
{
    return x + y;
}
```

A função executa com os valores passados e retorna o resultado



```
void main()
{
    float x = somar(4.5, 7.1);
}
```

A variável x recebe o resultado da função

Funções - Parâmetros

- Parâmetros são as informações necessárias para a execução de uma função
  - **Parâmetros formais** → parâmetros na declaração da função
  - **Parâmetros atuais** → parâmetros na chamada da função

```
float somar(float x, float y)
{
    return x + y;
}
```

```
void main()
{
    somar(4.5, 7.1);
}
```

Funções - Parâmetros

- Parâmetros são as informações necessárias para a execução de uma função
  - **Parâmetros formais** → parâmetros na declaração da função
  - **Parâmetros atuais** → parâmetros na chamada da função

```
float somar(float x, float y)
{
    return x + y;
}
```

Parâmetros formais

```
void main()
{
    somar(4.5, 7.1);
}
```

Parâmetros atuais

Funções - Exemplo

```
#include <stdio.h>

float somar(float x, float y)
{
    return x + y;
}

int main()
{
    float x = somar(4.5, 3.7);
    printf("O valor de x é: %.2f", x);
    return 0;
}
```

Funções - Modularização

- Um dos principais usos de funções é a modularização de um código, ou seja, dividir o código em partes menores cada uma com seu propósito (até em arquivos diferentes)
- Exibir um menu, pegar os dados de uma pessoa, indicar se um número é primo ou não...
- Sempre que determinada tarefa tiver que ser executada, é só executar o módulo referente a ela.

Funções - Exercício 1

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

void menu()
{
    \\imprimir menu
}
int ler_opcao()
{   \\perguntar opção
    return op;
}
```

```
int main()
{
    int op=-1;
    while(op!=0)
    {
        menu();
        op = ler_opcao();
        \\executa tarefa escolhida
    }
    return 0;
}
```

Complete este código com as opções básicas de uma calculadora

Funções - Exemplo

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

char perguntar()
{
    char resp;
    printf("Continuar? (s/n)");
    scanf(" %c", &resp);
    return resp;
}
```

```
int main()
{
    char resp='s';
    int cont=0;
    while (resp == 's')
    {
        printf("%d\n",++cont);
        resp = perguntar();
    }
    return 0;
}
```

Você consegue identificar o que esta função está fazendo?

Funções - Escopo de Variáveis

- O escopo de uma variável define em quais partes do programa ela existe e pode ser utilizada

```
int x=-5,a=4;
if (x>0)
{
    int b=a+2;
}
printf("%d", b);
```

Neste código, a variável b não existe fora do if, então há um erro de compilação indicando que a variável não foi declarada

Funções - Escopo de Variáveis

- O escopo de uma variável define em quais partes do programa ela existe e pode ser utilizada

Ou seja, o escopo da variável b é o bloco do if, então só existirá dentro deste bloco.

Ao terminar o escopo, a variável deixa de existir e o endereço de memória usado por ela é liberado.

```
int x=-5,a=4;
if (x>0)
{
    int b=a+2;
    printf("%d", b);
}
```

Funções - Escopo de Variáveis

- Em C, o programa tem dois tipos de ambiente
  - Global → externo às funções
  - Local → interno a uma função ou a uma estrutura de controle de fluxo

Funções - Escopo de Variáveis

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

//declaração de variáveis globais


// ---------------- Funçao main()------------------
int main(void)
{
  //declaração das variáveis locais da main()

   return(0);
}
// -------------------------------------------------


void funcao1(variáveis locais de parâmetros)
{
   // declaração das variáveis locais da função1

   return;
}
```

Funções - Escopo de Variáveis

- Uma variável declarada no ambiente global é chamada de variável global
    - Inicializadas automaticamente com 0
    - Podem ser usadas em qualquer função do programa
- Uma variável declarada em um ambiente local é chamada de variável local
    - Não é inicializada automaticamente com um valor (lixo de memória)
    - Só existem dentro da função ou estrutura

Funções - Passagem por referência

- Quando passamos uma variável como parâmetro para uma função, o valor da variável será copiado, então dizemos que temos uma **passagem de parâmetro por valor**.
- Dependendo de quantas vezes a função seja executada, pode-se levar a um esgotamento da memória já que cada vez que a função é chamada os valores são copiados na memória.
- Mesmo que tenha o mesmo nome, uma variável declarada dentro da função é diferente de uma variável da main()

Funções - Passagem por referência

- Exemplo

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

void trocar(int a, int b)
{
    int aux;
    aux = a;
    a = b;
    b = aux;
}
```

```
int main()
{
    int a=5, b=3;
    trocar(a,b);
    printf("%d %d", a, b);
}
```

Digamos que queremos uma função que troque os valores de duas variáveis

Funções - Passagem por referência

- Exemplo

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

void trocar(int a, int b)
{
    int aux;
    aux = a;
    a = b;
    b = aux;
}
```

```
int main()
{
    int a=5, b=3;
    trocar(a,b);
    printf("%d %d", a, b);
}
```

Esta função **trocar** não irá funcionar, pois as variáveis a e b da **main** são diferentes das variáveis a e b da função (como é uma passagem de parâmetro por valor, são cópias dos valores originais em outros endereços de memória)

Funções - Passagem por referência

- A **passagem de parâmetro por referência** consiste em passar o endereço de memória dos valores utilizados ao invés de fazer uma cópia desses valores.
- Com isso, qualquer operação pode ser realizada diretamente nos valores originais através do endereço de memória.

Funções - Passagem por referência

- Exemplo

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

void trocar(int a, int b)
{
    int aux;
    aux = a;
    a = b;
    b = aux;
}
```

```
int main()
{
    int a=5, b=3;
    trocar(a,b);
    printf("%d %d", a, b);
}
```

Vamos modificar esta função para uma passagem por referência.

Funções - Passagem por referência

- Exemplo

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

void trocar(int *a, int *b)
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

```
int main()
{
    int a=5, b=3;
    trocar(&a,&b);
    printf("%d %d", a, b);
}
```

Ao invés de receber valores inteiros,
vamos receber o endereço de memória,
ou seja, utilizaremos ponteiros.

Funções - Passagem por referência

- Exemplo

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

void trocar(int *a, int *b)
{
    int aux;
    aux = *a;
    *a = *b;
    *b = aux;
}
```

```
int main()
{
    int a=5, b=3;
    trocar(&a,&b);
    printf("%d %d", a, b);
}
```

Agora a função funciona corretamente, pois a troca de valores ocorre direto nos endereços de memória originais de a e b

Exercícios

- Faça uma função para informar se um número inteiro é maior ou menor que 0 e outra função para informar se um número é par ou ímpar.

- Escreva um programa que receba dois números e apresente as opções abaixo para o usuário. Utilize uma função que receba os dois números e retorne o maior deles e outra função que retorne o menor.

    - 1- Imprimir os números em ordem crescente
    - 2- Imprimir os números em ordem decrescente
    - 3- Imprimir o maior
    - 4- Imprimir o menor

Exercícios

- Faça uma função em C que receba 2 pares de coordenadas (x1, y1) e (x2,y2) e calcule a distância entre eles. A fórmula para o cálculo da distância entre dois pontos 2D é:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

#include <math.h>
sqrt(x) retorna a raiz quadrada de x
pow(x,2) retorna x^2

Exercícios

- Faça uma função que receba um valor inteiro e retorne 1 se este valor for primo ou 0 se não for primo.
- Faça uma função chamada **intRand** que receba dois valores inteiros (*a* e *b*) e retorne um valor inteiro aleatório entre eles.
- Faça uma função chamada **floatRand** que receba dois valores inteiros (*a* e *b*) e retorne um valor float aleatório entre eles com até 2 casas decimais

Exercícios

- Faça uma função **charRand** que retorne uma letra aleatória do alfabeto (desconsidere cedilha e acentos)
    - Dica: na tabela ASCII as letras maiúsculas vão do valor 65 (‘A’) ao 90 (‘Z’) e as minúsculas vão do valor 97 (‘a’) ao 122 (‘z’)
- Faça uma função que receba 3 valores inteiros (horas, minutos e segundos) e retorne o valor deste tempo convertido em segundos.
- Faça uma função que receba 2 valores inteiros positivos e retorne a soma dos números primos entre eles.