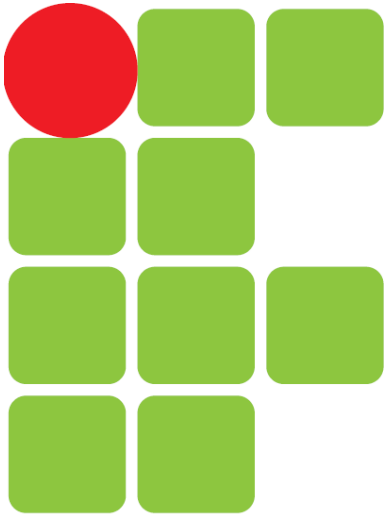

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

CAMPUS AFOGADOS DA INGAZEIRA

Introdução à Programação

Funções

Prof. Dr. Diego Rodrigues de Almeida
diego.rodrigues@afogados.ifpe.edu.br
https://sites.google.com/site/ifpediego

Motivação

- Mostrar exemplo de código
  - Ex1.c
  - Exemplo Banco Santos

Motivação

- E se quisermos dar desconto na compra acima de R$ 50,00???
  - Teremos que colocar um if em cada **case** do **switch**

- Devemos **REFATORAR** o código de forma a evitar a repetição de código
  - Refatorar é o ato de reorganizar o código de forma que ele fique mais simples e/ou rápido e seja equivalente ao código anterior

- Assim, para o código anterior teremos

Motivação

- Um importante recurso apresentado nas linguagens de programação é a modularização, onde um programa pode ser particionado em sub-rotinas bastante específicas.

- A linguagem C possibilita a modularização por meio de funções

- As funções possibilitam reaproveitar código e torna-lo mais simples

Funções

- Nós já conhecemos alguns que já vêm prontas em C
  - printf("Texto");
  - scanf("%i", &variavel);
  - gets(string);

- Mas C também permite que o programador crie suas próprias funções tornando seu código mais simples e fácil de ser entendido

Funções

- Uma função é definida da seguinte maneira:

```
tipo_de_dado_retornado nome_da_funcao(lista_de_parâmetros){
    Declaração de Variáveis
    Sequência de instruções
    return (se o tipo retornado não for void)
}
```

Tipo de dado retornado

- Uma função **pode** ou **não** retornar um valor
- Funções que não retornam valores retornam tipo **void**
- O tipo **void** simboliza que nada será retornado
- Para declarar uma função é preciso dizer explicitamente qual é o tipo do valor que será retornado
- Exemplo:
  - void
  - int
  - float
  - double
  - char
  - String

Nome da função

- O nome da função é o identificador da função
- Será utilizada quando o programa "chamar" a função
- Um programa "chama" uma função pelo seu nome
- Quando um programa "chama" uma função, ele deseja que a função execute e retorne o seu resultado (caso a função retorne algum valor)
- Valem as mesmas regras de identificadores de variáveis

Lista de parâmetros

- Para realizar uma tarefa muitas vezes uma função precisa de parâmetros

- Os parâmetros são as variáveis ou vetores que ele necessita para realizar sua tarefa

Lista de parâmetros

- Não há uma quantidade mínima e máxima de parâmetros a serem passados
- A declaração dos parâmetros seguem a forma geral:

```
tipo_de_dado_retornado nome_da_funcao(tipo1 p1, tipo2 p2, ..., tipoN pN){
    Declaração de Variáveis
    Sequência de instruções
    return (se o tipo retornado não for void)
}
```

- Onde tipo 1 é o tipo do parâmetro p1, o tipo 2 é o tipo do parâmetro p2 e assim sucessivamente

Return

- O return determina o que a função retornará

- A função só executa até o return

Exemplo

- Vamos refatorar o exemplo Ex1.c

Exemplo

- Programa para calcular o fatorial de um número natural

```
int main() {
   setlocale(LC_ALL, "Portuguese");
   int numero, fatorial;
   imprime_boas_vindas();
   numero = le_numero_valido();
   fatorial = calcula_fatorial(numero);
   imprime_resultado(numero, fatorial);
   return 0;
}
```

Exemplo

```
void imprime_boas_vindas(){
   printf(" --------------------------Bem vindo--------------------------\n");
   printf("|                                                              |\n");
   printf("| Este programa calcula o fatorial de um número natural |\n");
   printf("|                                                              |\n");
   printf(" --------------------------Bem vindo--------------------------\n");
}
```

```
int main() {
   setlocale(LC_ALL, "Portuguese");
   int numero, fatorial;
   imprime_boas_vindas();
   numero = le_numero_valido();
   fatorial = calcula_fatorial(numero);
   imprime_resultado(numero, fatorial);
   return 0;
}
```

Exemplo

```
int le_numero_valido(){
   int numero;
   do{
      printf("Digite um número natural: ");
      scanf("%i", &numero);
   }while(numero < 0);
   return numero;
}
```

```
int main() {
   setlocale(LC_ALL, "Portuguese");
   int numero, fatorial;
   imprime_boas_vindas();
   numero = le_numero_valido();
   fatorial = calcula_fatorial(numero);
   imprime_resultado(numero, fatorial);
   return 0;
}
```

Exemplo

```
int calcula_fatorial(int numero){
   int fatorial = 1;
   for(int i = numero; i > 0; i--){
      fatorial = fatorial * i;
   }
   return fatorial;
}
```

```
int main() {
   setlocale(LC_ALL, "Portuguese");
   int numero, fatorial;
   imprime_boas_vindas();
   numero = le_numero_valido();
   fatorial = calcula_fatorial(numero);
   imprime_resultado(numero, fatorial);
   return 0;
}
```

Exemplo


```
void imprime_resultado(int numero, int fatorial){
   printf("O fatorial de %i é %i\n", numero, fatorial);
}
```

```
int main() {
   setlocale(LC_ALL, "Portuguese");
   int numero, fatorial;
   imprime_boas_vindas();
   numero = le_numero_valido();
   fatorial = calcula_fatorial(numero);
   imprime_resultado(numero, fatorial);
   return 0;
}
```

Arquivo header (.h)

- É possível organizar funções em um arquivo header com a extensão .h e fazer a inclusão no arquivo que desejar.

- Mostrar exemplo

Exercício

- Crie uma função que recebe um número inteiro e retorna seu valor multiplicado por 100

Exercício

- Utilizando funções, faça um programa que o usuário digita um número e o programa diz se é ímpar ou par

- O programa deve ter:
  - Uma função para ler o número
  - Uma função que dado o número retorna true se for par ou false se for ímpar

Exercício

- Crie uma função que receba 3 números e retorne o maior valor

Passagem por Cópia ou por Referência

- Em C, a passagem de parâmetros pode ser feita por valor (cópia) ou por referência (ponteiro).

- Vamos ver cada uma dessas abordagens com exemplos:

Passagem por Valor (Cópia)

- Quando você passa um argumento por valor, a função recebe uma cópia do valor do argumento.
- Qualquer modificação feita no parâmetro dentro da função não afeta a variável original.

```
// Função que incrementa o valor de x
void incrementaPorValor(int x) {
    x = x + 1; // Isso não afeta a variável original
    printf("Dentro da função (por valor): x = %d\n", x);
}

int main() {
    int a = 5;
    printf("Antes de chamar a função: a = %d\n", a);
    incrementaPorValor(a);
    printf("Depois de chamar a função: a = %d\n", a); // a permanece 5
    return 0;
}
```

Passagem por Referência (Ponteiro)

- Quando você passa um argumento por referência, a função recebe um ponteiro para a variável. Isso permite que a função modifique a variável original.

```
// Função que incrementa o valor de x usando um ponteiro
void incrementaPorReferencia(int *x) {
    *x = *x + 1; // Isso afeta a variável original
    printf("Dentro da função (por referência): x = %d\n", *x);
}

int main() {
    int a = 5;
    printf("Antes de chamar a função: a = %d\n", a);
    incrementaPorReferencia(&a); // Passa o endereço de a
    printf("Depois de chamar a função: a = %d\n", a); // a agora é 6
    return 0;
}
```

Passagem por Referência (Ponteiro)

- Em C, argumentos do tipo vetor são modificados quando são passados como parâmetro para funções

```
void imprimeVetor(int vetor[], int tamanho){
   for(int i = 0; i < tamanho; i++){
      printf("%i ", vetor[i]);
   }
   printf("\n");
}

void zeraVetor(int vetor[], int tamanho){
   for(int i = 0; i < tamanho; i++){
      vetor[i] = 0;
   }
}

int main() {
    int vetor[] = {1, 2, 3, 4, 5};
    printf("Antes de chamar a função: vetor = ");
    imprimeVetor(vetor, 5);//Antes de motificar
    zeraVetor(vetor, 5);
    printf("Depois de chamar a função: vetor = ");
    imprimeVetor(vetor, 5);//Depois de motificar
    return 0;
}
```

Exercício

- Crie uma função que recebe um vetor e o inverte

Exercício

- Crie uma função que uma string e retorna o seu tamanho

Escopo de Variáveis (Variáveis Locais)

- Variáveis declaradas no interior de uma função só são acessíveis por instruções nesta função.

- Na realidade, elas só existem durante a execução da função: são "criadas" quando a função é ativada e são "destruídas" quando termina a execução da função.

- Entende-se por escopo o bloco em que o a variável é acessível

Escopo de Variáveis (Variáveis Locais)

- Qual vai ser o resultado na tela da execução desse programa?

```
void funcao1() {
   int numero = 2;
   printf("2 - numero = %i\n", numero);
}

void funcao2(int numero) {
   numero++;
   printf("4 - numero = %i\n", numero);
}

int main() {
   int numero = 1;
   printf("1 - numero = %i\n", numero);
   funcao1();
   printf("3 - numero = %i\n", numero);
   funcao2(numero);
   printf("5 - numero = %i\n", numero);
   return 0;
}
```

Escopo de Variáveis (Variáveis Locais)

- Qual vai ser o resultado na tela da execução desse programa?

```
void funcao1() {
   int numero = 2;
   printf("2 - numero = %i\n", numero);
}

void funcao2(int numero) {
   numero++;
   printf("4 - numero = %i\n", numero);
}

int main() {
   int numero = 1;
   printf("1 - numero = %i\n", numero);
   funcao1();
   printf("3 - numero = %i\n", numero);
   funcao2(numero);
   printf("5 - numero = %i\n", numero);
   return 0;
}
```

RESPOSTA

1 – numero = 1
2 – numero = 2
3 – numero = 1
4 – numero = 2
5 – numero = 1

Escopo de Variáveis (Variáveis Globais)

- Se uma variável deve ser acessada por mais de uma função, ela deve ser declarada fora de qualquer função, sendo chamada, neste caso, de *variável global*.

- Uma variável global pode ser referenciada em qualquer função do programa

Escopo de Variáveis (Variáveis Globais)

- Qual vai ser o resultado na tela da execução desse programa?

```
int numero;

void funcao1() {
   numero = 2;
   printf("2 - numero = %i\n", numero);
}

void funcao2(int numero) {
   numero++;
   printf("4 - numero = %i\n", numero);
}

int main() {
   numero = 1;
   printf("1 - numero = %i\n", numero);
   funcao1();
   printf("3 - numero = %i\n", numero);
   funcao2(numero);
   printf("5 - numero = %i\n", numero);
   return 0;
}
```

Escopo de Variáveis (Variáveis Globais)

- Qual vai ser o resultado na tela da execução desse programa?

```
int numero;

void funcao1() {
   numero = 2;
   printf("2 - numero = %i\n", numero);
}

void funcao2(int numero) {
   numero++;
   printf("4 - numero = %i\n", numero);
}

int main() {
   numero = 1;
   printf("1 - numero = %i\n", numero);
   funcao1();
   printf("3 - numero = %i\n", numero);
   funcao2(numero);
   printf("5 - numero = %i\n", numero);
   return 0;
}
```

RESPOSTA

1 – numero = 1
2 – numero = 2
3 – numero = 2
4 – numero = 3
5 – numero = 2

Escopo de Variáveis (Variáveis Globais)

- Pode-se identificar uma variável local com o mesmo identificador de uma variável global

- Neste caso, referências ao identificador comum dentro da função no qual a variável local foi definida refere-se a esta variável local

- Isto, naturalmente, impede a função de acessar a variável global.

Escopo de Variáveis (Variáveis Locais)

- Qual vai ser o resultado na tela da execução desse programa?

```
int numero;

void funcao1() {
   int numero = 2;
   printf("2 - numero = %i\n", numero);
}

void funcao2(int numero) {
   numero++;
   printf("4 - numero = %i\n", numero);
}

int main() {
   numero = 1;
   printf("1 - numero = %i\n", numero);
   funcao1();
   printf("3 - numero = %i\n", numero);
   funcao2(numero);
   printf("5 - numero = %i\n", numero);
   return 0;
}
```

Escopo de Variáveis (Variáveis Locais)

- Qual vai ser o resultado na tela da execução desse programa?

```
int numero;

void funcao1() {
   int numero = 2;
   printf("2 - numero = %i\n", numero);
}

void funcao2(int numero) {
   numero++;
   printf("4 - numero = %i\n", numero);
}

int main() {
   numero = 1;
   printf("1 - numero = %i\n", numero);
   funcao1();
   printf("3 - numero = %i\n", numero);
   funcao2(numero);
   printf("5 - numero = %i\n", numero);
   return 0;
}
```

RESPOSTA

1 – numero = 1
2 – numero = 2
3 – numero = 1
4 – numero = 2
5 – numero = 1

Obrigado