Linguagem C

Funções

Professora: Emanoeli Madalosso
emanoelim@utfpr.edu.br

Funções

- Até o momento vimos que um um programa é composto por uma função principal, a função **main()**. Todo o código que queremos executar é escrito dentro dessa função;
- Também vimos como utilizar algumas funções prontas provenientes de bibliotecas como a math.h ou a string.h:
  - **sqrt(numero)** -> retorna a raiz quadrada do número informado
  - **strlen(string)** -> retorna o tamanho da string informada
- Também podemos criar nossas próprias funções, conforme a nossa necessidade.

Funções

- O que é uma função?
  - É um subprograma, ou um conjunto de instruções, que tem como objetivo resolver um problema específico.
  - Esse conjunto de instruções fica agrupado sob um identificador, tendo a seguinte estrutura:

Funções

- Estrutura geral:

```
tipo_do_retorno identificador(parâmetro ou lista de parâmetros) {
        // conjunto de instruções
}
```

cabeçalho

corpo

identificador: nome da função;

parâmetro ou lista de parâmetros: os dados que a função precisa para realizar o processamento:

- Cada dado está associado a um tipo;
- Se mais de um parâmetro for usado, eles devem ser separados por vírgulas.

tipo de retorno: após fazer o processamento, a função pode devolver um valor, é necessário especificar seu tipo (int, float, char, etc).

Criação de uma função

- Exemplo: função para somar dois números inteiros:

```
int soma_dois_numeros(int numero1, int numero2) {
    int resposta = numero1 + numero2;
    return resposta;
}
```

  - o **identificador** da função deve representar com clareza o que ela faz;
  - todos os **parâmetros** estão precedidos por seus tipos e estão separados por vírgulas;
  - a palavra reservada **return** permite que a função devolva a resposta calculada;
  - como estamos somando dois inteiros, a resposta da função será do tipo inteiro, por isso precisamos definir o **tipo do retorno** como int.

Criação de uma função

- Em que lugar do código a função deve ser criada?
  - O compilador deve conhecer o código da função antes de ela ser usada na main. Isso pode ser feito de 3 formas:
    - Criando a função antes da função main;
    - Criando a função depois da função main;
    - Criando a função em arquivo separado.

Criação de uma função

- Antes da função main:

```c
#include <stdio.h>

int soma_dois_numeros(int n1, int n2) {
    int resposta;
    resposta = n1 + n2;
    return resposta;
}
main() {
    int n1, n2, resposta;
    printf("Informe dois números para somar: ");
    scanf("%d%d", &n1, &n2);
    resposta = soma_dois_numeros(n1, n2);
    printf("%d", resposta);
}
```

Como a função foi criada antes da main, o compilador já conhece o código da função antes de ela ser usada.

Criação de uma função

- Depois da função main:
  - Ao criar uma função após a main, para que ela seja reconhecida pelo compilador, é preciso criar um **protótipo** da função antes da main.
  - Protótipos de função tem o propósito de informar ao compilador a existência da função.
  - O protótipo de uma função corresponde ao seu cabeçalho.

Criação de uma função

- Antes da função main:

```c
#include <stdio.h>

int soma_dois_numeros(int n1, int n2);

main() {
    int n1, n2, resposta;
    printf("Informe dois números para somar: ");
    scanf("%d%d", &n1, &n2);
    resposta = soma_dois_numeros(n1, n2);
    printf("%d", resposta);
}
int soma_dois_numeros(int n1, int n2) {
    int resposta;
    resposta = n1 + n2;
    return resposta;
}
```

O compilador conhece o código da função antes de ela ser usada, pois seu protótipo está antes da main.

Não devemos esquecer o “ponto e vírgula” após o protótipo.

Criação de uma função

- Em arquivo separado:
  - A função é criada em um arquivo com extensão “.h” (de *header file*);
  - Este tipo de arquivo contém funções que podem ser usadas por diversos programas;
  - Existem dois tipos de arquivo .h:
    - Aqueles que vem com o compilador: stdio.h, stdlib.h, string.h, time.h, etc.
    - Aqueles criados pelo programador.

Criação de uma função

- Em arquivo separado:
  - Arquivos .h padrões do compilador e arquivos .h criados pelo programador são incluídos em um programa usando a diretiva de pré-processamento “include”.
  - Para os arquivos padrão, o nome do arquivo deve estar entre <>. Isso indica que o arquivo deve ser procurado nos diretórios padrão do sistema:
    - #include <string.h>
  - Para arquivos criados pelo programador, deve-se colocar o nome do arquivo entre aspas duplas. Isso indica que o arquivo será procurado no diretório onde se encontra o programa:
    - #include “minhas_funcoes.h”

Criação de uma função

- Em arquivo separado:
  - A função soma_dois_numeros é criada em um arquivo chamado “funcoes.h”:

```
int soma_dois_numeros(int n1, int n2) {
    int resposta;
    resposta = n1 + n2;
    return resposta;
}
```

Criação de uma função

- Em arquivo separado:
  - No programa principal incluimos o arquivo “funcoes.h”:

```
#include <stdio.h>
#include “funcoes.h”

main() {
    int n1, n2, resposta;
    printf("Informe dois números para somar: ");
    scanf("%d%d", &n1, &n2);
    resposta = soma_dois_numeros(n1, n2);
    printf("%d", resposta);
}
```

Chamada de função

- Como utilizar a função soma_dois_numeros?
  - Através de uma **chamada de função**:

```c
#include <stdio.h>
#include “funcoes.h”

main() {
    int n1, n2, resposta;
    printf("Informe dois números para somar: ");
    scanf("%d%d", &n1, &n2);
    resposta = soma_dois_numeros(n1, n2);
    printf("%d", resposta);
}
```

Como a função possui um valor retorno, é preciso atribuir este retorno à uma variável

Na chamada da função devem ser passados todos os parâmetros necessários, como especificado no cabeçacho da função

Chamada de função

- Atenção:

  chamada: **resposta = soma_dois_numeros(n1, n2);**

  cabeçalho: **int soma_dois_numeros(int n1, int n2)**

  - O cabeçalho indica que a função espera 2 valores inteiros, por isso na chamada deve-se passar exatamente 2 valores inteiros (se forem passados valores float, por exemplo, não ocorre erro, mas serão consideradas apenas as partes inteiras dos valores).
  - Ao passar os parâmetros na chamada da função, passamos apenas os nomes das variáveis que guardam os valores, ou os próprios valores. Não colocamos os tipos antes dos nomes.
  - Os nomes das variáveis que são passadas para a função não precisam coincidir com os nomes usados na função.

Tipos de funções

- Funções podem ser classificadas em relação ao retorno e aos parâmetros:
  - Função com parâmetros e com retorno;
  - Função com parâmetros e sem retorno;
  - Função sem parâmetros e com retorno;
  - Função sem parâmetros e sem retorno;

Função com parâmetros e com retorno

- A função recebe um ou mais parâmetros, calcula um valor e devolve esse valor.
- Para uma função devolver um valor usamos o comando **return**.
- Sempre que uma função que tem retorno é chamada na main, deve-se atribuir o resultado da função para uma variável do mesmo tipo do retorno.
- ATENÇÃO: uma função consegue retornar apenas UM valor.

Função com parâmetros e com retorno

A função vai retornar um número inteiro

A função recebe como parâmetro um número inteiro

```c
#include <stdio.h>

int verifica_par(int n);

main() {
    int n, resposta;
    printf("Informe um numero para verificar: ");
    scanf("%d", &n);
    resposta = verifica_par(n);
    if (resposta == 1)
        printf("Seu numero eh par.");
    else
        printf("Seu numero eh impar.");
}

int verifica_par(int n) {
    if(n % 2 == 0)
        return 1;
    else
        return 0;
}
```

Como esta função tem retorno, é necessário atribuir à uma variável

Função com parâmetros e com retorno

- O comando **return**:
  - Quando é encontrado um comando return, a execução da função é imediatamente finalizada e volta a ser executada a função chamadora (a main, por ex.);
  - Não são executados quaisquer outros comandos da função após a execução de um comando return.

Função com parâmetros e com retorno

- O comando **return**:
  - O comando return pode estar acompanhado de uma variável:
    - return x;
    - return(x);

    equivalentes

    Neste caso, além de voltar o fluxo para a função chamadora, a função chamadora recebe o valor x;
  - Pode estar sozinho:
    - return;

    Neste caso, somente volta o fluxo para a função chamadora.

Função com parâmetros e sem retorno

- Bastante utilizada para realizar impressão de valores na tela;
- Como a função não tem retorno, ao chamar ela na main, não é preciso atribuir a uma variável;
- Exemplo: uma função que imprime a tabuada de um determinado número:

Função com parâmetros e sem retorno

A palavra reservada **void** indica que a função não tem retorno

```c
#include <stdio.h>

void imprime_tabuada(int n);

main() {
    int n;
    printf("Informe um numero para mostrar sua tabuada: ");
    scanf("%d", &n);
    imprime_tabuada(n);
}

void imprime_tabuada(int n) {
    int i;
    for(i=1; i<=10; i++)
        printf("%d * %d = %d\n", i, n, i*n);
}
```

A função recebe como parâmetro um número inteiro

Como esta função não tem retorno, não é necessário atribuir à uma variável

Função sem parâmetros e com retorno

- Neste caso a função retorna um valor mas não precisa de parâmetros.
- Neste caso chamamos:
  - **variavel = nome_funcao();** // apenas com os () sem nada dentro
- Um exemplo é uma função que gera um número aleatório:

Função sem parâmetros e com retorno

A palavra reservada **int** indica que a função vai retornar um número inteiro

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

int gera_numero_aleatorio(void);

main() {
    int numero = gera_numero_aleatorio();
    printf("Numero: %d", numero);
}

int gera_numero_aleatorio(void) {
    srand(time(NULL));
    return rand() % 100;
}
```

A palavra reservada **void** indica que a função não precisa de parâmetros

Como esta função tem retorno, é necessário atribuir à uma variável

Função sem parâmetros e sem retorno

- Neste caso a função não retorna nenhuma resposta e também não precisa de nenhuma entrada. Um exemplo é uma simples função que escreve um texto na tela:

A palavra reservada **void** indica que a função não tem retorno

```c
#include <stdio.h>

void imprime_msg_boas_vindas(void);

main() {
    imprime_msg_boas_vindas();
}

void imprime_msg_boas_vindas(void) {
    printf("Bem vindo(a)!");
}
```

A palavra reservada **void** indica que a função não precisa de parâmetros

Como esta função não tem retorno, não é necessário atribuir à uma variável

Escopo de variáveis

- O **escopo** de uma variável está associado à sua visibilidade dentro do programa, que pode ser:
  - **Local**: a variável é conhecida somente pela função na qual está declarada.
  - **Global**: a variável é conhecida por todas as funções do programa.

Escopo local

```c
#include <stdio.h>

int verifica_par(int n);

main() {
    int i, resposta;
    for(i=1; i<=10; i++) {
        resposta = verifica_par(i);
        if(resposta == 1)
            printf("%d\t", i);
    }
}

int verifica_par(int n) {
    if(n % 2 == 0)
        return 1;
    else
        return 0;
}
```

A variável **i** foi declarada dentro da função *main* e é conhecida somente dentro dela. Para a função *verifica_par* a variável **i** não existe. Logo:

- Se tentarmos imprimir a variável **i** dentro da função *verifica_par*, um erro será exibido:
  error :“i” undeclared.
- Se tentarmos declarar uma variável chamada **i** dentro da função *verifica_par*, não acontecerá problema de redeclaração, pois do ponto de vista da função *verifica_par*, a variável **i** ainda não existe.

Escopo global

- Para uma variável ter escopo global ela deve estar declarada no início do programa, fora de qualquer função. Assim, ela será conhecida e **compartilhada** por todas as funções.
- Deve-se ter muito cuidado ao usar variáveis globais, pois se uma função altera uma variável global, a alteração feita se estenderá para todas as funções que utilizam essa variável.

Escopo global

```c
#include <stdio.h>

int calcula_potencia(int exp);

int base = 2;

main() {
    int expoente;
    printf("Informe o expoente: ");
    scanf("%d", &expoente);
    printf("%d^%d = %d", base, expoente, calcula_potencia(expoente));
}

int calcula_potencia(int exp) {
   int i, resposta = base;
   for(i=1; i<exp; i++)
       resposta *= base;
   return resposta;
}
```

A variável base tem escopo global, sendo conhecida tanto pela função *main* quanto pela função *calcula_potencia*.

Escopo global

- Qual é o problema do código abaixo?

```c
#include <stdio.h>

void imprime_tabuada(int n);
int i; // variável global

main() {
    for(i=1; i<=10; i++) {
        imprime_tabuada(i);
        printf("\n");
    }
}

void imprime_tabuada(int n) {
   for(i=1; i<=10; i++)
       printf("%d * %d = %d\n", i, n, i*n);
}
```

Obs. 1

- Quando você passa um valor como parâmetro para uma função, o que a função recebe é uma CÓPIA desse valor;
- Portanto, qualquer alteração feita, terá efeito nesta cópia e não sobre o valor original.
- Exemplo:

Obs. 1

```c
#include <stdio.h>

int quadrado(int numero) {
    numero *= numero;
    return numero;
}

int main(void) {
    int numero = 10;
    int resposta = quadrado(numero);
    printf("Resposta: %d\n", resposta);
    printf("Número original: %d\n", numero);
}
```

Ao imprimir a variável "numero" na função main, ela ainda está com o valor 10

Vantagens de utilizar funções

- Quando quebramos um programa grande em subprogramas menores, o código fica mais fácil de entender, pois cada conjunto de instruções com um objetivo específico fica agrupado sob um identificador que indica o que o conjunto de instruções está fazendo.
- Reutilização de código: imagine um programa onde você precisa imprimir várias matrizes. Em vez de escrever a lógica para imprimir uma matriz várias vezes, você pode criar uma função chamada *imprimir_matriz* e chamar esta função quantas vezes precisar.
- Facilita a realização de testes: cada funcionalidade pode ser testada separadamente, facilitando na hora de encontrar qual funcionalidade está com erro. Fundamental em TDD (*Test Driven Development*).

Obs. 2

- Para maior clareza do código e para garantir a reutilização de código, uma função deve:
  - Ter um nome claro que especifique exatamente o que a função faz (mesmo que fique um pouco extenso);
  - Uma função deve ter APENAS UM OBJETIVO. Uma função nunca deve fazer duas coisas diferentes. Exemplo: se for necessário calcular a média de idades em um vetor, achar maior idade e achar a menor idade, não crie uma função que faz essas 3 coisas. Crie uma função separada para cada coisa.