

FUNÇÕES EM C

Professor: Francisco Dantas Nobre Neto
E-mail: dantas.nobre@ifpb.edu.br

Agenda

2

- Introdução
- Funções
- Passagem de Parâmetros
- Argumentos de Programas
- Referências Bibliográficas

Introdução

3

- Programar em C significa criar funções que possuem os blocos de código a serem executados:
 - ▣ O programa principal de C é a função **main()**.
- Uma função executa um trecho de código e, geralmente, retorna um valor;
- Agrupar trechos do programa em funções tem diversas vantagens:
 - ▣ Melhora a legibilidade;
 - ▣ Facilita o reuso de código;
 - ▣ Evita duplicação de esforço.

Introdução

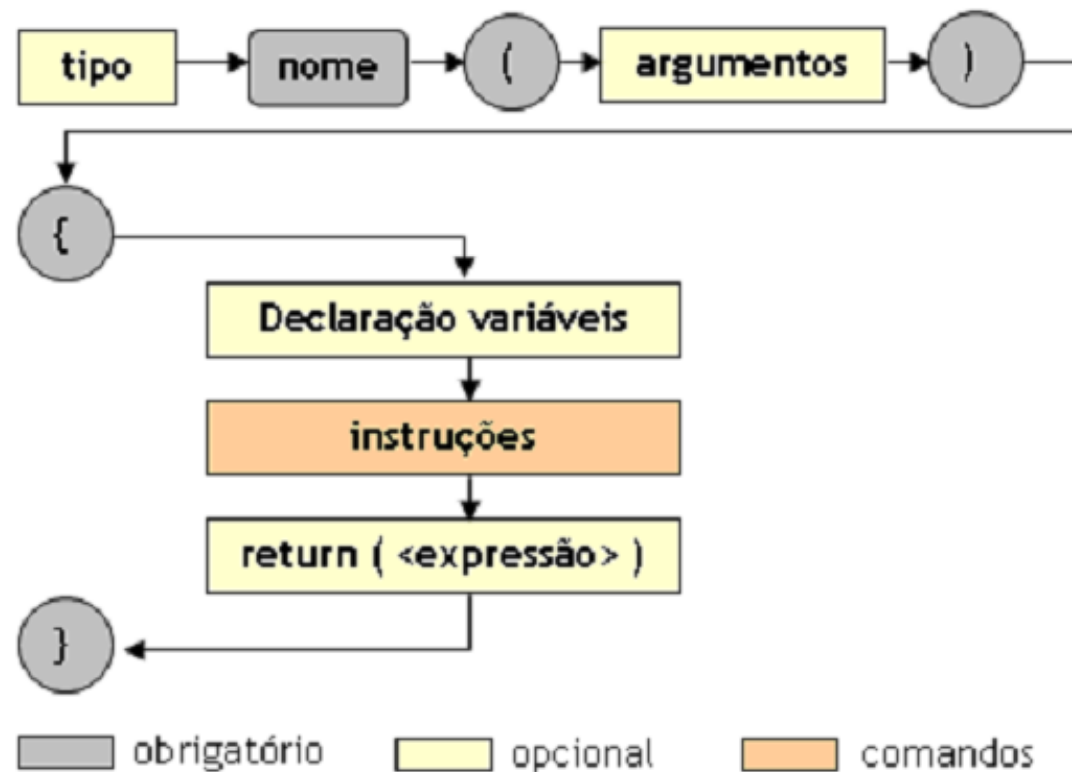
4

- É possível passar informações extras para as funções poderem executar corretamente:
 - ▣ São os argumentos (parâmetros) da função.
- Exemplos de funções com parâmetros:
 - ▣ `scanf("%d", &quantidade);`
 - ▣ `printf("quantidade de aluno %d", quantidade);`

Introdução

5

- A estrutura de uma função é a seguinte:



Funções

6

- O *tipo* da função não é obrigatório:
 - ▣ Se for omitido, será admitido o tipo *int* (default).
- É possível utilizar o tipo *void*:
 - ▣ Significa não precisar retornar valor, por parte da função.
- Uma chamada a uma função **desvia** o fluxo de execução do programa para o endereço correspondente dela.

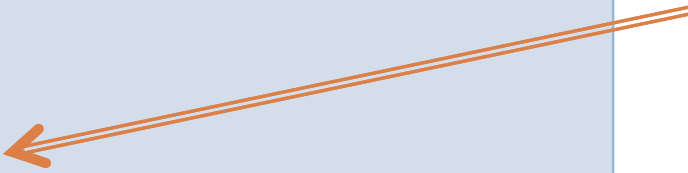
Funções

7

□ Exemplo.

```
int mensagem(){  
    printf("Seja bem-vindo!");  
    return (0);  
}  
  
int main(){  
    mensagem();  
    printf("Já foi escrito alguma coisa na tela!");  
    return (0);  
}
```

É obrigatório chamar a função com os parênteses, mesmo que não precise passar parâmetros!!!



Funções

8

□ Comando **return**:

- Saída imediata da função que o contém;
- Devolver um valor para o trecho de código que o chamou (exceto para o tipo **void**);
- É possível ter vários *return* dentro de uma função:
 - Porém, não é boa prática.
- Se um *return* não estiver presente na função, seu valor de retorno é indefinido.
- O tipo de retorno pode ser:
 - Tipos primitivos: char, int, float, double, long int, long double;
 - Estruturas definidas pelo programador;
 - Um ponteiro.

Funções

9

- Função que retorna um número inteiro.

```
int soma(int a, int b){  
    return (a + b);  
}  
  
int main(){  
    int x;  
    x = soma(2, 3);  
    printf("A soma dos números 2 e 3 é igual a %d", x);  
    return (0);  
}
```


Funções

10


□ Considerações:

- ▣ Variáveis declaradas na função são **locais a ela**;
- ▣ Todas as funções estão no mesmo nível de escopo;
- ▣ Os argumentos usados para chamar a função devem ser compatíveis com o tipo de seus parâmetros:

```
int soma(int a, int b){  
    return (a + b);  
}  
  
int main(){  
    int x = soma(2, 3);  
    printf("Soma de 2 e 3 é %d", x);  
    return (0);  
}
```



```
int soma(int a, int b){  
    return (a + b);  
}  
  
int main(){  
    int x = soma("2", "3");  
    printf("Soma de 2 e 3 é %d", x);  
    return (0);  
}
```



Funções

11

- Por que usar funções?
 - ▣ Evita duplicação de esforço (código e correção de erros);
 - ▣ Reaproveitamento de código;
 - ▣ Para evitar um trecho de código muito longo, que dificulta a legibilidade;
 - ▣ Para agrupar um trecho de código que possui uma semântica correlacionada.

Funções

12

- Declaração de função:
 - ▣ Toda função deve ser declarada antes de ser usada:
 - Geralmente, elas são usadas dentro da função principal **main()**.
 - ▣ Para isso, escreve-se apenas o protótipo da função, antes da função **main()**:
 - Apenas a declaração é feita no início;
 - A função é definida após a função **main()**.

Funções

13

□ Declaração da função.

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

/* O protótipo das funções fica depois
da inclusão das bibliotecas */
int soma(int a, int b);

int main(){
    int x = soma(2, 3);
    printf("Soma de 2 e 3 é %d", x);
    return (0);
}

int soma(int a, int b){
    return (a+b);
}
```

Funções

14

- Conceitualmente, uma função sempre retorna um valor;
- Em C, só temos função, não temos como declarar uma *procedure* (procedimento);
- Como fazer que função, em C, se comporte como um procedimento???

Utilizando um tipo de
retorno especial,
void

Funções

15

- Com o tipo de retorno **void**, a função não precisa retornar valor:
 - ▣ Mas é possível usar o **return**, para encerrar o fluxo de execução.

```
#include <stdio.h>
void soma(int a, int b);

int main(){
    soma(2, 3);
    return (0);
}

void soma(int a, int b){
    if(b > a) return;
    printf("Valor da soma: ", (a+b));
}
```

Passagem de Parâmetros

16

- Parâmetros formais:
 - ▣ São os parâmetros definidos na declaração da função.
- Parâmetros reais:
 - ▣ São os valores ou variáveis passados na chamada da função.
- Regras:
 - ▣ A quantidade de parâmetros reais tem que ser igual à de parâmetros formais;
 - ▣ Os parâmetros (reais e formais) têm que ser compatíveis.

Passagem de Parâmetros

17

□ Parâmetros formais x Parâmetros reais.



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

```
/* Protótipo das funções */
```

```
int soma(int a, int b);
```

```
int main(){
```

```
    printf("Soma é %d", soma(2, 3));
```

```
    return (0);
```

```
}
```

```
int soma(int a, int b){
```

```
    return (a+b);
```

```
}
```

Parâmetros reais



X

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

```
/* Protótipo das funções */
```

```
int soma(int a, int b);
```

```
int main(){
```

```
    printf("Soma é %d", soma(2));
```

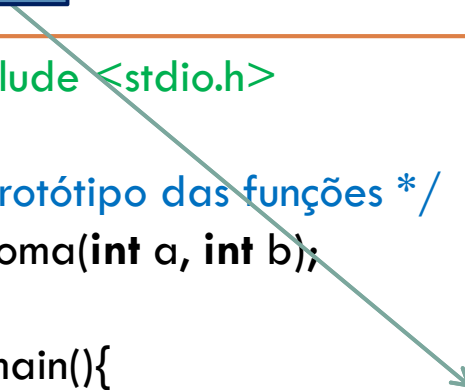
```
    return (0);
```

```
}
```

```
int soma(int a, int b){
```

```
    return (a+b);
```

```
}
```



Parâmetros formais

Passagem de Parâmetros

18

- Na linguagem C, é possível passar parâmetros de duas formas:
 - ▣ Passagem por valor:
 - Os parâmetros reais são **copiados** para serem usados dentro da função.
 - ▣ Passagem por referência:
 - É passada a referência (posição em memória) da variável.

Passagem de Parâmetros

19

□ Passagem por valor:

- ▣ Qualquer alteração da variável informada (parâmetro real) para a função chamada, não será refletida na função chamadora.

```
int main(){
    int x = 2, y = 9;
    troca(x, y);
    printf("Valores trocados: %d - %d", x, y);
    return (0);
}

void troca(int a, int b){
    int temp = a;
    a = b;
    b = temp;
}
```

Passagem de Parâmetros

20

- Passagem por referência:
 - ▣ Dentro da função chamada, qualquer alteração do parâmetro formal, será refletido no parâmetro real;
 - ▣ Não é feito uma cópia dos valores, mas informada a sua posição em memória;
 - ▣ Em C, usa-se apontadores para passar argumentos por referência.

Passagem de Parâmetros

21

□ Passagem por referência.

```
int main(){  
    int x = 2, y = 9;  
    troca(&x, &y);  
    printf("Valores trocados: %d - %d", x, y);  
    return (0);  
}  
  
void troca(int *a, int *b){  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

Serão passadas as posições de memórias (referências) das variáveis!

Recebe uma posição de memória, cujo conteúdo desta posição é um inteiro!

Passagem de Parâmetros

22

□ Exemplo:

- ▣ Faça uma função e um procedimento para calcular o quadrado de um número.

```
int f_quadrado(int num);
void p_quadrado(int * num);

int main(){
    int num = 2; p_quadrado(&num);
    printf("Valor: %d", f_quadrado(num)); // O que será impresso???
    return (0);
}

int f_quadrado(int num){
    return (num * num);
}

void p_quadrado(int *num){
    *num = *num * *num;
}
```

Passagem de Parâmetros

23

- ❑ Passando vetores como parâmetro:
 - ❑ Vetores são passados, sempre, como referência:
 - Qualquer alteração em um dos elementos do vetor, será refletido na função chamadora.
 - ❑ É passada a posição de memória do primeiro elemento do vetor, para a função.

```
void imprime(int* v);

int main(){
    ...
}

void imprime(int* v){
    int i;
    for(i = 0; i < 3; i++) printf("%d", v[i]);
}
```



```
void imprime(int v[]);

int main(){
    int vetor[] = {5, 8, 10};
    imprime(vetor);
    return (0);
}

void imprime(int v[]){
    int i;
    for(i = 0; i < 3; i++) printf("%d", v[i]);
}
```

Argumentos de Programas

24

- É possível passar argumentos para o programa no momento de sua chamada;
- Para isso, existem dois parâmetros na função **main()**:
 - ▣ Um é o *argc*:
 - É um inteiro;
 - Recebe o número de argumentos passado.
 - ▣ O outro é o *argv*:
 - É um vetor para *strings*;
 - Armazena ponteiros que apontam para cada um dos argumentos passados.

Argumentos de Programas

25

```
#include <stdio.h>

/* Recebendo parâmetros no execução do
   programa */
int main(int argc, char *argv[]){
    int cont;
    for(cont = 0; cont < argc; cont++){
        printf("%s ", argv[cont]);
    }
}
```

Após compilar esse arquivo fonte, deve-se chamar o arquivo executável gerado, e passar os parâmetros nesse momento!

MS-DOS:

```
C:\Dev-Cpp\bin>gcc -ansi ..\ws\aula.c
```

```
C:\Dev-Cpp\bin>aula param1 param2 param3
```

argc = 3

argv[0] = "param1"

argv[1] = "param2"

argv[2] = "param3"

Referências Bibliográficas

26

- Curso de Linguagem C, UFMG:
 - ▣ http://www.ead.cpdee.ufmg.br/cursos/C/Programa_C.pdf

- Uso de Funções em C, PUC-RS, Professor Márcio Sarroglia Pinho:
 - ▣ <http://www.inf.pucrs.br/~pinho/Laprol/Funcoes/AulaDeFuncoes.htm>