parte5 _ FUNÇÕES

Marcos Silvano / DACOM

BCC32A-Algoritmos 1

Roteiro

- Visão geral
- Criando funções
- Chamando funções
- Passagem de parâmetros & retorno
  - Escopo de variáveis
  - “Compondo” funções
- Múltiplos pontos de retorno
- Próximos
  - Parâmetros por referência
  - Pilha de chamada de funções

Funções: definição e uso

Funções

- Uma função agrupa linhas para formar uma funcionalidade
  - ***"Mini programa"***
- Escrevemos funções para:
  - Reutilizar código
  - Modularizar código
- Em outras palavras: criar blocos de código com funcionalidades que podem ser reusadas
- Sintaxe de uso (chamada) de função:

```
nome_funcao()
nome_funcao(parametros)
```

- Sintaxe da declaração de função:

```
tipo_retorno nome_funcao (paremetros) {
    corpo_da_funcao (bloco_de_codigo)
    retorno
}
```

Funções em nosso código

- Na verdade, sempre estivemos escrevendo uma função: **main()**

```c
#include <stdio.h>

int main() { // tipo_retorno + nome_funcao
    // bloco de codigo/corpo da funcao
    int a;
    printf("Entre com um valor:\n"); // chamada
    scanf("%d", &a);                 // chamada
    printf("Numero digitado: %d\n"); // chamada
    // retorno
    return 0;
}
```

- E também utilizamos funções, desde o início:
  - biblioteca <stdio.h>: printf() e scanf()
  - biblioteca <math.h>: sqrt()
- Similaridade com matematica é (bastante) breve:

```
f(x) = x * x/2
```

Criando e chamando funções

- Vamos considerar o código abaixo:

```c
#include <stdio.h>

int main() {
    printf("+-----------+\n");
    printf("|           |\n");
    printf("|           |\n");
    printf("|           |\n");
    printf("+-----------+\n");

    return 0;
}
```

- Cada vez que desejar imprimir uma nova caixa, preciso **duplicar esse código**
  - Solução mais inteligente -> transformá-lo em um bloco de código reutilizável

Criando e chamando funções

- Vamos considerar o código abaixo:

```c
#include <stdio.h>

void printBox() { // void = sem retorno
    printf("+-----------+\n");
    printf("|           |\n");
    printf("|           |\n");
    printf("|           |\n");
    printf("+-----------+\n");
}

int main() {
    printBox();
    printBox();

    return 0;
}
```

- Funções ajudam a padronizar comportamentos no código
  - Se precisar alterar, faremos somente em um único local
  - Ex: vamos alterar o tamanho da caixa

Passando e obtendo valores de uma função

Passagem de parâmetros

- Declaramos parâmetros quando desejamos enviar dados à função
  - Também chamados de **parâmetros de entrada**

```c
#include <stdio.h>
// parâmetros funcionam como variáveis e recebem cópias dos valores passados
void printSomatorio(int a) {
    int som = 0;
    while (a > 0) {
        som += a;
        a--;
    }
    printf("Somatorio de %d: %d\n", a, som);
}
int main() {
    printSomatorio(5); // parâmetro a=5
    printSomatorio(10)); // parâmetro a=10

    return 0;
}
```

Retorno de função

- Para retornar um valor por uma função, devemos:
  - Informar o **tipo** a retornar na declaração da função
  - Encerrar e função e retornar o valor com o comando **"return"**
- Só é possível retornar um único valor em uma função
  - Para retornar mais de um valor em uma função, utilizamos **passagem de parâmetros por referência**

```c
#include <stdio.h>
int somatorio(int a) { // função retornará "int"
    int som = 0;
    while (a > 0) {
        som += a;
        a--;
    }
    return som;
}
int main() {
    int s = somatorio(5);
    printf("Somatorio de 5: %d\n", s);
    // possível utilizar o retorno de uma função diretamente
    printf("Somatorio de 10: %d\n", somatorio(10));
    return 0;
}
```

Múltiplos pontos de retorno

- Podemos incluir mais pontos de retorno (“**return**”)
- Caso comum: tratamento de erros

```c
#include <stdio.h>

/* Considere uma funcao que divide somente naturais.
   Retorno < 0 significa erro.
*/
float div(int a, int b) {
    if (b == 0) {
        printf("\nERRO: div(%d,%d) denominador não pode ser zero.\n");
        return -1;
    }
    if (a < 0 || b < 0) {
        printf("\nERRO: div(%d,%d) não permitidos inteiros negativos.\n");
        return -1;
    }
    return a/b;
}
int main() {
    printf("Div 5/2: %.1f\n", div(5,2));
    printf("Div -5/2: %.1f\n", div(-5,2));
    printf("Div 3/0: %.1f\n", div(3,0));
    return 0;
}
```

Passagem de parâmetros a funções

Passagem de parâmetros: cópia vs referência 1/3

- Como pudemos anteriormente, usamos parâmetros para passar valores à função. Isso é chamado de passagem de **parâmetros por cópia**
  - Os valores passados são copiados para os parâmetros da função

```c
void printMedia(int a, int b, int c) {
    int media = (a + b + c) / 3;
    printf("Media entre %d, %d e %d: %d\n", a, b, c, media);
}

int main() {
    // Os valores 5, 10 e 15 são copiados para os parâmeros da função,
    // na mesma ordem: a = 5, b = 10 e c = 15
    printMedia(5, 10, 15);

    // De forma similar, a = 30, b = 60 e c = 90
    int x = 30, y = 60, z = 90;
    printMedia(x, y, z);

    return 0;
}
```

Passagem de parâmetros: cópia vs referência 2/3

- A passagem de parâmetros a funções pode ser por
  - **Cópia**: considerados parâmetros de entrada de dados.
  - **Referência**: considerados parâmetros de entrada ou saída de dados.
- Passarem de **parâmetros por referência**
  - O parâmetro faz uma referência (aponta) para a variável externa, podendo alterá-la

```
// PASSAGEM DE PARÂMETROS POR CÓPIA
// Alterar os parâmetros não afeta as variáveis externas, pois os parâmetros
// apenas recebem cópias dos valores
void fun_copia(int a, int b) {
    a = 100;
    b = 200;
}

// PASSAGEM DE PARÂMETROS POR REFERÊNCIA
// É possível alterar as variáveis externas, pois os parâmetros as referenciam
void fun_referencia(int* a, int* b) {
    *a = 100;
    *b = 200;
}
```

Passagem de parâmetros: cópia vs referência 3/3

- O código abaixo utiliza as funções **fun_copia** e **fun_referencia** do slide anterior.

```
int main() {
    // Os valores das varáveis m e n são copiados para os parâmetros a e b,
    // respectivamente. Alterar a e b internamente à função não afeta m e n.
    int m = 5, n = 10;
    fun_copia(5, 10, 15);

    prinf("%d %d\n", m, n); // imprime 5 10

    // x e y serão alterados pela função, uma vez que os parâmetros a e b passam
    // a referenciar (apontar) as variáveis externas x e y
    int x = 5, y = 10;
    // é necessário utilizar o operador & para indicar que estamos passando
    // o endereço da variável (ao invés do valor)
    fun_referencia(&x, &y);

    prinf("%d %d\n", x, y); // imprime 100 e 200

    return 0;
}
```

Escopo e emprego de funções

Escopo de variáveis de funções

- Em qual trecho de código as variáveis **a** e **som** são visíveis?

```c
#include <stdio.h>

int somatorio(int a) { // <- variável a
    int som = 0;       // <- variável som
    while (a > 0) {
        som += a;
        a--;
    }
    return som;
}
int main() {
    int s = somatorio(5);
    printf("Somatorio de 5: %d\n", s);
    printf("---------------------------\n");

    printf("Somatorio de 10: %d\n", somatorio(10));
    printf("---------------------------\n");
    return 0;
}
```

“Compondo” funções

- Repare que estamos repetindo o texto sublinhado.
  - Vamos transformá-lo em uma **unidade reutilizável**

```c
#include <stdio.h>

int somatorio(int a) {
    int som = 0;
    while (a > 0) {
        som += a;
        a--;
    }
    return som;
}
int main() {
    printf("Somatorio de 5: %d\n", somatorio(5));
    printf("---------------------------\n");

    printf("Somatorio de 10: %d\n", somatorio(10));
    printf("---------------------------\n");

    printf("Somatorio de 36: %d\n", somatorio(36));
    printf("---------------------------\n");
    return 0;
}
```

“Compondo” funções

- Repare que estamos repetindo o texto sublinhado.
  - Vamos transformá-lo em uma **unidade reutilizável**

```c
#include <stdio.h>

int somatorio(int a) {
    int som = 0;
    while (a > 0) {
        som += a;
        a--;
    }
    return som;
}
void printSomatorio(int a) {
    // a função printSomatorio() utiliza a função somatorio()
    printf("Somatorio de %d: %d\n", a, somatorio(a));
    printf("----------------------------\n");
}
int main() {
    printSomatorio(5);
    printSomatorio(10);
    printSomatorio(36);
    return 0;
}
```