UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Funções

Aula 06

Marcos Silvano Almeida
marcossilvano@professores.utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

Funções

- Uma função é um **subprograma**
  - Sequência de comandos para realizar uma tarefa específica
  - Pequenos módulos de programa que favorecem a reutilização de código
  - Escondem a complexidade da implementação: como funciona o printf?
- Sintaxe de uso da função (chamada):
  - `nome_da_função(param1, param2, ...)`

```c
// função que imprime na saída padrão (terminal)
printf("Exemplo de saida: %d e %d\n", a, b);

// função que lê da entrada padrão (terminal)
scanf("%d %d", &a, &b);

// retorna um número aletarório entre 0 e 2.147.483.647 (RAND_MAX)
int sorteio = rand() % 10; // 0 a 9
```

As funções estão contidas nas bibliotecas

- As funções que utilizamos até o momento estão escritas em bibliotecas
  - As bibliotecas são escritas na linguagem C
    - Usam a linguagem para criar funcionalidades reutilizáveis
- As bibliotecas contém
  - Valores padrões (macros): INT_MAX, INT_MIN, RAND_MAX, ...
  - Funções: printf(), scanf(), rand(), sqrt(), ...
  - Tipos customizados e estruturados (em outra aula)

```c
printf("Resultado: %d e %d\n", a, b);          // biblioteca <stdio.h>

scanf("%d %d", &a, &b);                        // biblioteca <stdio.h>

int sorteio = rand() % 10; // 0 a 9            // biblioteca <stdlib.h>
```

UTFPR

Chamando funções: parâmetros e retorno

- Funções podem receber parâmetros para o seu funcionamento

```
printf("Resultado: %d e %d\n", a, b);
scanf("%d %d", &a, &b);
```

- Funções podem retornar um valor ao finalizarem

```
// printf retorna a quantidade de caracteres escritos na saída
int qte = printf("Resultado: %d e %d\n", a, b);
int sorteio = rand() % 10; // 0 a 9
```

- Quando uma função é chamada, o **código chamador** fica parado na linha da função e só volta a executar quando a **função finalizar**

Criando funções

Um primeiro exemplo: função para imprimir texto

```c
#include <stdio.h>
/*
   Sintaxe:
     tipo_do_retorno nome_da_função(parâmetros)
       void = não retorna valor
*/
void print_data() {
   printf("\nPrimeira funcao!\n");
   printf("-----------------\n\n");
}

int main() {
   print_data();
   return 0;
}
```

Parâmetros para a função

```c
#include <stdio.h>
/*
  Sintaxe:
    Parâmetros na declaração são separados por vírgula
*/
void print_data(int a, float b) {
   printf("\nDados informados\n");
   printf("-----------------\n");
   printf("  int..: %d\n", a);
   printf("  float: %.2f\n\n", b);
}

int main() {
   print_data(5, 45.0732);
   return 0;
}
```

Atividades

(1) Escreva uma função que imprime:

```
--------------------
  HELLO C FUNCTION
--------------------
```

(2) Escreva uma função que recebe três valores (float) e imprime sua média aritmética simples.

Declaração completa: parâmetros + retorno

- Um programa C inicia pela função main()
  - Estávamos escrevendo uma função em todos os programas: main()
- Sintaxe completa da declaração de uma função

```
tipo_retorno nome_da_função(int param1, int param2,...) {
    ...
    return valor; // se houver tipo de retorno
}
```

- Observe a declaração da função main()

```
int main() {
    // conteúdo...
    return 0; // deve retornar int
}
```

Return define os pontos de saída da função.

tipo_retorno = int, float, char...
Encerra a função e retorna valor
(return é obrigatório)

tipo_retorno = void
Encerra a função mais cedo
(opcional para void)

Função para devolver o maior entre dois valores

```c
int max(int a, int b) {
    int maior;
    if (a > b) {
        maior = a;
    } else {
        maior = b;
    }
    return maior;
}
```

```c
int main() {
    int x = 5, y = 10;

    int res = max(x, y);
    printf("MAIOR: %d\n", res);

    printf("Max: %d\n", max(x, y));

    printf("Max: %d\n", max(924, 127));

    printf("Max: %d\n", max(90, max(15, 12)) );
    return 0;
}
```

Função para devolver o maior entre dois valores

```c
/* Versão mais curta:
     Usa retornos antecipados.

   Podemos utilizar retornos antecipados
     quando for necessário/conveniente
 */
int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

```c
int main() {
    int x = 5, y = 10;

    int res = max(x, y);
    printf("MAIOR: %d\n", res);

    printf("Max: %d\n", max(x, y));

    printf("Max: %d\n", max(924, 127));

    printf("Max: %d\n", max(90, max(15, 12)) );
    return 0;
}
```

Definindo protótipos de funções

- A função deve estar declarada antes (acima) de ser utilizada.
  - É possível definir o protótipo das funções antes de implementá-las.

```c
#include <stdio.h>

// protótipos/assinaturas
int max(int a, int b);
void maiorEntreTres();

int main() {
    maiorDeTres();
    return 0;
}
```

```c
void maiorDeTres() {
    int a, b, c;
    printf("Informe 3 valores: ");
    scanf("%d %d %d", &a, &b, &c);
    printf("MAX:%d\n", max(max(a,b), c));
}
int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```

Atividades

(3) Função que retorna a média aritmética de 3 valores

(4) Função que retorna um conceito A, B ou C, para um valor n:

a. C, se 0 <= n < 60
b. B, se 60 <= n < 80
c. A, se 80 <= n <= 100

Passagem de parâmetros para funções

Passagem de parâmetro: exemplo mais elaborado

⇒ **Por valor**: parâmetro de entrada

- A função recebe uma cópia do valor passado aos parâmetros.

```c
// Verifica se número é primo (true/false)
int isPrime(int num) {
    if (num % 2 == 0) { // divide por 2?
        return 0;
    }
    int div = 3; // verif apenas impares
    while (div < sqrt(num)) {
        if (num % div == 0) {
            return 0; // false
        }
        div += 2;
    }
    return 1; // true
}
```

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

int main() {
    printf("Prime: %d\n", isPrime(561));
    printf("Prime: %d\n", isPrime(1021));

    return 0;
}
```

Passagem de parâmetro por endereço

⇐⇒ **Por endereço**: parâmetro de entrada e saída

- A função recebe um parâmetro com o endereço da variáveil passada à função

```c
// Recebe três números e retorna o maior e o menor
void minMax(int a, int b, int c,
            int* min, int* max) {
    int menor = a;
    if (b < menor) menor = b;
    if (c < menor) menor = c;

    int maior = a;
    if (b > maior) maior = b;
    if (c > maior) maior = c;

    *min = menor;
    *max = maior;
}
```

```c
#include <stdio.h>

int main() {
    int x,y,z;
    printf("Informe 3 numeros: ");
    scanf("%d %d %d", &x, &y, &z);

    int m, n;
    minMax(x, y, z, &n, &m);
    printf("Menor: %d\n", n);
    printf("Maior: %d\n", m);
    return 0;
}
```

Passagem de parâmetro por endereço

⇐⇒ **Por endereço**: parâmetro de entrada e saída

- A função recebe um parâmetro com o endereço da variáveil passada à função

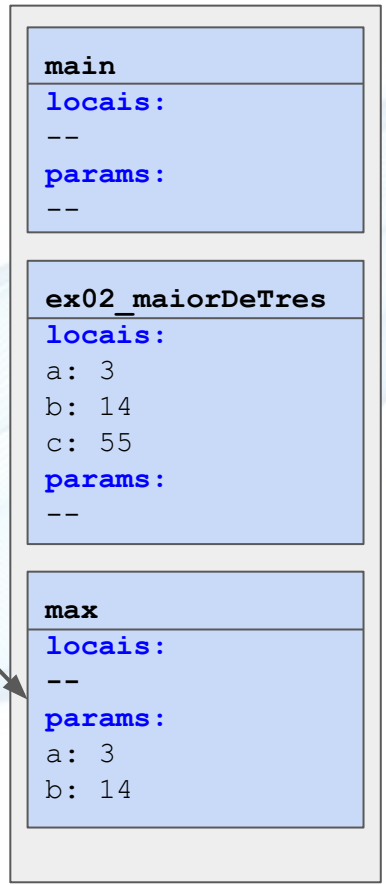
```c
// Calcula as raízes do polinômio 2º grau
// Forma: ax² + bx + c = 0
int polyRoots2(float a, float b, float c,
               float *root1, float *root2) {
    // 1º grau: bx + c = 0
    if (a == 0)
        *root1 = *root2 = -c / b;
    float delta = b * b - 4 * a * c;
    if (delta < 0)
        return 0; // não possui raízes reais
    *root1 = (-b - sqrt(delta)) / (2 * a);
    *root2 = (-b + sqrt(delta)) / (2 * a);
    return 1;
}
```

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

int main() {
    float r1 = 0;
    float r2 = 0;

    polyRoots2(4, -10, 4, &r1, &r2);

    printf("Roots: %0.2f, %0.2f\n", r1, r2);
    return 0;
}
```

Pilha de Chamada de Funções

A pilha de chamadas de funções

```c
int max(int a, int b) {
    if (a > b) return a;
    else       return b;
}
void maiorDeTres() {
    int a, b, c;
    printf("Informe 3 valores: ");
    scanf("%d %d %d", &a, &b, &c);

    printf("MAX:%d\n", max(max(a,b), c));
}
int main() {
    maiorDeTres();
    return 0;
}
```



Adendo: problema de leitura de char

Adendo: problema de leitura de caracteres

- **scanf(“%c”)** consome um caractere do buffer de entrada
  - Quando digitamos um caractere e pressionamos ENTER, este último ficará no buffer de entrada como ‘\n’, que alimentará automaticamente o próximo scanf(“%c”), impedindo a leitura de dois caracteres em sequência.

- Duas soluções
  - **SOLUÇÃO 1: scanf(“ %c”) ⇒ espaço + %c**
    - O espaço antes do %c indica que o separador de entrada deve ser ignorado
  - **SOLUÇÃO 2: limpar buffer de entrada após leitura**
    - Utilizar scanf(“%c”) e chamar uma função para consumir o restante do buffer até ‘\n’

```c
// utilizar após scanf ou getchar
void clearBuffer() {
    while (getchar() != '\n');
}
```

Referências

- Algoritmos e Programação
  - Marcela Gonçalves dos Santos
  - Disponível pelo Moodle

- Estruturas de Dados, Waldemar Celes e José Lucas Rangel
  - PUC-RIO - Curso de Engenharia
  - Disponível pelo Moodle

- Linguagem C, Silvio do Lago Pereira
  - USP - Instituto de Matemática e Estatística
  - Disponível pelo Moodle

- Curso Interativo da Linguagem C
  - https://www.tutorialspoint.com/cprogramming