Funções

Disciplina de Programação de Computadores I
Universidade Federal de Ouro Preto

Agenda

Sub-rotinas: Função e Procedimentos
A função main
Variáveis globais e locais
Escopo de variáveis
Passagem de Parâmetros por Valor
Protótipos de sub-rotinas

Funções

- Frequentemente, em algumas situações, dividimos um problema maior em problemas menores e resolvemos os esses problemas.

- Ao criarmos um programa para resolver um problema, utilizamos funções para codificar trechos do programa que resolvem problemas menores.

- As funções devem codificar a solução para um problema pequeno e específico.

Por que utilizar Funções?

- Evitar que os programas fiquem grandes demais e difíceis de serem lidos e compreendidos.

- Separar o programa em partes que possam ser compreendidas de forma isolada (criação de módulos).

- Utilizar um código em diferentes partes do programa, sem que ele precise ser escrito em cada local em que se deseje utilizá-lo.

- Permitir o reuso de código em outros programas (bibliotecas).

Declaração de funções

Função Nome (Dados de Entrada)

Comandos

Dados de Saída

fim

Exemplo de declaração de funções

```
Função soma (inteiro X, inteiro Y)

 inteiro Z

  Z =  (X+Y)

  retorne Z

fim
```

Exemplo de declaração de funções

```
Função soma (inteiro X, inteiro Y)

 inteiro Z

 Z =  (X+Y)

 retorne Z

Fim

inicio
 inteiro A, B, C
 leia(A,B)
 C = soma(A, B)
 escreva(C)
fim
```

A	B	C	X	Y	Z
5	3	8	5	3	8

Exemplo de declaração de funções

```
Função soma (inteiro X, inteiro Y)
  inteiro Z
  Z =  (X+Y)
  escreva (Z)
fim
```

Exemplo de declaração de funções

Função soma (inteiro X, inteiro Y)

 inteiro Z

 Z = (X+Y)

 escreva Z

Fim

A	B	X	Y	Z

inicio
 inteiro A, B
 leia(A,B)
 soma(A, B);
fim

Declaração de funções em C

```
tipo_retorno nome ( tipo parâmetro1, ..., tipo parâmetroN)

{
      comandos;
      return variável_tipo_retorno;
}
```

Exemplo de função em C com retorno de valor

```
#include<stdio.h>

int soma (int X, int Y) {

    return (X+Y);
}

int main( ) {

    int A, B, C;
    scanf("%d %d", &A, &B);
    C = soma(A, B);
    printf("\n%d", C);

    C = soma(2, 2);
    printf("\n%d", C);

    C = soma(3, 10);
    printf("\n%d", C);

    C = soma(20, 5);
    printf("\n%d", C);

    return 0;
}
```

Exemplo de função em C sem retorno de valor

```
#include<stdio.h>

void soma (int X, int Y) {

    printf("\n%d",X+Y);
}

int main( void ) {

    int A, B;
    scanf("%d %d", &A, &B);

    soma(A, B);
    soma(2, 2);
    soma(3, 10);
    soma(20, 5);

    return 0;
}
```

O tipo *void*

- void é um tipo especial que indica “nada” ou “vazio”.

- void é utilizado para indicar que:
  - Uma função não retorna
  - Uma função possui uma lista vazia de parâmetros.

Regras para definições de Funções

- Funções só podem ser declaradas fora de outras funções.
- Funções devem ser declaradas **antes** de serem usadas.
- O uso de **void** na lista de parâmetros na declaração de funções é opcional, mas indicado por clareza.
- O tipo de retorno de uma função não declarado é assumido ser **int**.

A função main

- A função **main** é a primeira função executada no programa.

- Ela possui tipo de retorno fixo (**int**) e é chamada automaticamente pelo sistema operacional quando o programa é executado.

- O comando **return**, neste caso, indica ao sistema operacional se o programa funcionou corretamente ou não.

- Por padrão, o valor **0** é interpretado como funcionamento correto e demais valores são interpretados como erros.

Declarações comuns da função main

```
int main (void) {... return 0;}
```

Este programa não recebe parâmetros na linha de comando.

```
int main (int argc, char const * argv[]) {... return 0;}
```

Este programa recebe parâmetros na linha de comando.

- argc indica a quantidade de parâmetros; e
- argv[] permite recuperar os parâmetros

Exemplo 1 – Função com retorno de valor

```
Função celsius_fahrenheit ( real tc )

    real tf
    tf = 1.8 * tc + 32

    retorne tf

fim

inicio

    real cels, fahr;

    escreva(Entre com temperatura em Celsius: )
    leia(cels)

    fahr = celsius_fahrenheit(cels)

    escreva( fahr)


fim
```

Exemplo 1 – Função com retorno de Valor

```
#include <stdio.h>

float celsius_fahrenheit ( float tc );

int main() {

    float cels, fahr;

    printf("Entre com temperatura em Celsius: ");
    scanf("%f", &cels);

    fahr = celsius_fahrenheit(cels);

    printf("Temperatura em Fahrenheit: %f”, fahr);

    return 0;
}
float celsius_fahrenheit ( float tc ){
    float tf;
    tf = 1.8 * tc + 32;
    return tf;
}
```

Exemplo 1 – Função sem retorno de Valor

```
Função celsius_fahrenheit ( real tc )

    real tf
    tf = 1.8 * tc + 32

    escreva( tf)

fim
```

```
inicio

    real cels,;

    escreva(Entre com temperatura em Celsius: )
    leia(cels)

    celsius_fahrenheit(cels)

fim
```

Exemplo 1 – Função sem retorno de Valor

```
#include <stdio.h>

void celsius_fahrenheit ( float tc );

int main() {

    float cels;

    printf("Entre com temperatura em Celsius: ");
    scanf("%f", &cels);

    celsius_fahrenheit(cels);

    return 0;
}
void celsius_fahrenheit ( float tc ){

    float tf;

    tf = 1.8 * tc + 32;
    printf("Temperatura em Fahrenheit: %f”, tf);
}
```

Exemplo 2 – Função com retorno de Valor

```
função quadrado (inteiro x)
    retorne  x * x
fim

inicio

    inteiro numero, res;

    escreva("Digite um numero inteiro: ");
    leia(numero);

    res = quadrado(numero);

    escreva(res);

fim
```

Exemplo 2 – Função com retorno de Valor

```
int quadrado (int x){
    return (x * x);
}

int main (){

int numero, res;

printf("Digite um numero inteiro: ");
scanf("%d", &numero);

res = quadrado(numero);

printf("O quadrado de %d eh %d.\n", numero, res);

return 0;

}
```

Exemplo 2 – Função sem retorno de Valor

```
função quadrado (inteiro x)
    escreva(x * x)
fim


inicio
    inteiro numero;

    escreva("Digite um numero inteiro: ");
    leia(numero);

    quadrado(numero);

fim
```

Exemplo 2 – Função sem retorno de Valor

```
void quadrado (int x){

    printf("O quadrado de %d eh %d.\n", x, x*x);
}

int main (){

    int numero;

    printf("Digite um numero inteiro: ");
    scanf("%d", &numero);

    quadrado(numero);

    return 0;

}
```

Exemplo 3 – Função com retorno de Valor

```
Função calculaMedia(real nota1, real nota2, real nota3)

    real media
    media = (nota1 + nota2 + nota3) / 3
    retorne media
fim
```

```
inicio
    real n1, n2, n3,res;
    escreva("Digite as 3 notas: ")
    leia(n1, n2, n3);
    res = calculaMedia (n1, n2, n3)
    imprima(res)
fim
```

Exemplo 3 – Função com retorno de Valor

```
float calculaMedia(float nota1, float nota2, float nota3){
    float media;
    media = (nota1 + nota2 + nota3) / 3;
    return media;
}
int main(){
    float n1, n2, n3,media;
    printf("Digite as 3 notas: ");
    scanf("%f %f %f", &n1, &n2, &n3);
    media = calculaMedia (n1, n2, n3);
    printf("Media = %.2f\n", media);
    return 0;
}
```

Exemplo 3 – Função sem retorno de Valor

```
Função calculaMedia(real nota1, real nota2, real nota3)

    real media
    media = (nota1 + nota2 + nota3) / 3
    imprima(media)
fim
```

```
inicio
    real n1, n2, n3;
    escreva("Digite as 3 notas: ")
    leia(n1, n2, n3);
    calculaMedia (n1, n2, n3)
fim
```

Exemplo 3 – Função sem retorno de Valor

```
void calculaMedia(float nota1, float nota2, float nota3)
{
    float media;
    media = (nota1 + nota2 + nota3) / 3;
    printf("Media = %.2f\n", media);
}

int main()
{
    float n1, n2, n3;
    printf("Digite as 3 notas: ");
    scanf("%f %f %f", &n1, &n2, &n3);
    calculaMedia (n1, n2, n3);
    return 0;
}
```

Variáveis globais e variáveis locais

- Variáveis declaradas fora de funções são chamadas globais e são visíveis a partir do ponto de declaração pelo restante do programa.

- Variáveis declaradas dentro de sub-rotinas são chamadas locais e só são visíveis dentro da sub-rotina em que foram declaradas.

- São variáveis locais:
  - variáveis declaradas dentro da sub-rotina
  - os parâmetros declarados na definição da sub-rotina

Exemplo de variáveis locais e globais

```
#include<stdio.h>

int contador_global=0;                    // Variável global


void imprimeMaior (int X, int Y) {        // X e Y são variáveis locais
    if (X > Y) printf(”%d”, X);           ao procedimento
    else printf(”%d”, Y);
}

int main() {
    int X, Y;
    scanf(”%d %d”, &X, &Y);               // X e Y são variáveis locais
    imprimeMaior(X, Y);                   à função, diferentes das
    return 0;                             variáveis do procedimento
}                                         anterior
```

Escopo de Variáveis

- O escopo de uma variável determina em quais partes do código ela pode ser acessada.

- Uma variável só pode ser acessada após o ponto em que é declarada.

- As regras de escopo de variáveis em C são:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

Exemplo de escopo de variáveis

```
#include<stdio.h>
int contador_global=0;                    // pode ser acessada em qualquer ponto do programa

void imprimeMaior (int X, int Y) {        // X e Y são visíveis apenas neste procedimento
     if (X > Y) printf(”%d”, X);
     else printf(”%d”, Y);
}

int main() {                              // A e B são visíveis apenas na função main
     int A, B;
     scanf(”%d %d”, &A, &B);
     imprimeMaior(A, B);
     return 0;
}
```

Passagem de Parâmetros por Valor

- Quando invocamos uma sub-rotina devemos fornecer, para cada um dos seus parâmetros, um valor de mesmo tipo do parâmetro, respeitando a ordem e a quantidade de parâmetros declarados.

- Ao invocarmos uma sub-rotina passando variáveis no lugar dos parâmetros, os valores das variáveis são **copiados** para os parâmetros da função.

- Alterações (dentro da sub-rotina) no valor dos parâmetros **não afetam** as variáveis usadas na chamada da função.

- Isto é chamado **Passagem de Parâmetros por Valor**

Protótipos (ou Assinaturas) de sub-rotinas (I)

- Para podermos implementar sub-rotinas em partes distintas do aquivo-fonte e podermos implementar sub-rotinas **depois** de utilizá-las, utilizamos protótipos (ou assinaturas) de sub-rotinas

- Protótipos correspondem à primeira linha da definição de uma função.

- O protótipo de uma sub-rotinas deve aparecer antes do uso desta sub-rotina.

- Em geral, colocam-se os protótipos no início do arquivo-fonte.

Protótipos (ou Assinaturas) de sub-rotinas (II)

```
tipo_retorno nome ( tipo parâmetro1, ..., tipo parâmetroN)
```

Assinatura (Protótipo) da função

```
{
comandos;
return variável_tipo_retorno;
}
```

Corpo (definição) da função

```
void nome ( tipo parâmetro1, ..., tipo parâmetroN)
```

Assinatura (Protótipo) do procedimento

```
{
comandos;
}
```

Corpo (definição) do procedimento

Definindo função após o uso através de protótipo

```
#include<stdio.h>

int soma (int X, int Y) ; // Assinatura

int main( void ) {
    int A, B, C;
    scanf(”%d %d”, &A, &B);
    C = soma(A, B); // Chamada da função
    printf(”%d”, C);
    return 0;
}

int soma (int X, int Y) { // Definição da função
    return (X+Y);
}
```

Referências Bibliográficas

Material de aula do Prof. Ricardo Anido, da UNICAMP:
http://www.ic.unicamp.br/~ranido/mc102/

Material de aula da Profa. Virgínia F. Mota:
https://sites.google.com/site/virginiaferm/home/disciplinas

DEITEL, P; DEITEL, H. *C How to Program*. 6a Ed. Pearson, 2010.