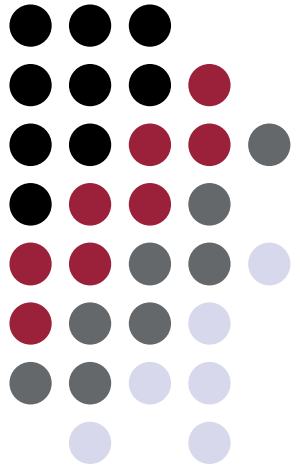
A06 Sub-rotinas

CSI030 – Programação de Computadores I

Prof. Dr. George H. G. Fonseca
Universidade Federal de Ouro Preto



Agenda

- Sub-rotinas: função e procedimentos
- A função main
- Variáveis locais e globais
- Escopo de variáveis
- Passagem de parâmetros por valor/referência
- Assinaturas de sub-rotinas

Sub-rotinas

- Frequentemente, dividimos um problema maior em problemas menores para serem resolvidos
- Ao criarmos um programa para resolver um problema, utilizamos sub-rotinas para codificar trechos do programa que resolvem problemas menores.
- As sub-rotinas devem codificar a solução para um problema pequeno e específico.
- Sub-rotinas podem ser funções ou procedimentos (quando não retornam valores).

Por que utilizar sub-rotinas?

- 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

```
tipo_retorno nome(tipo parametro1, ..., tipo parametroN)
{
    comando1;
    ...
    comandoN;
    return variavel_tipo_retorno;
}
```

- Uma função executa comandos e retorna algum resultado, cujo tipo é determinado por **`tipo_retorno`**.
- Cada parâmetro é uma variável que assume o valor que foi passado na chamada da função.
- O comando **`return`** fornece o resultado da execução dessa função para quem a chamou.

Declaração de funções

- Exemplo de função que soma dois inteiros

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  int soma (int x, int y)
 5  {
 6      return (x+y);
 7  }
 8
 9  int main( void )
10  {
11      int a, b, c;
12      scanf("%d %d", &a, &b);
13      c = soma(a, b);
14      printf("%d", c);
15      return 0;
16  }
```

Declaração de procedimentos

```
void nome(tipo parametro1, ..., tipo parametroN)
{
    comando1;
    ...
    comandoN;
}
```

- Um procedimento é um tipo especial de função que executa comandos e não retorna um resultado.
- O resultado do procedimento são as alterações executadas no estado do programa.
- Cada parâmetro é uma variável que assume o valor que for passado na chamada do procedimento.

O tipo void

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

- void é utilizado para indicar que:
  - Uma função não retorna valor (ou seja, que a função é um procedimento).

  - Uma função possui uma lista vazia de parâmetros.

Declaração de procedimentos

- Exemplo de procedimento que imprime o maior de dois números:

```
 1  #include <stdio.h>
 2
 3  void imprimeMaior (int x, int y) {
 4      if(x > y)
 5          printf("%d\n", x);
 6      else
 7          printf("%d\n", y);
 8  }
 9
10  int main( void ) {
11      int x, y;
12      scanf("%d %d", &x, &y);
13      imprimeMaior(x, y);
14      return 0;
15  }
```

Regras para a definição de sub-rotinas

- Sub-rotinas só podem ser declaradas fora de outras funções.
- Sub-rotinas 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 sub-rotina 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 retorno 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.

Variáveis globais e 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

Variáveis globais e locais

Exemplo

```
1  #include<stdio.h>
2
3  int contador_global=0; //variável global
4
5  void imprimeMaior (int x, int y) {
6      //x e y são variáveis locais do procedimento
7      if (x > y)
8          printf("%d", x);
9      else
10         printf("%d", y);
11 }
12
13 int main() {
14     /*x e y são variáveis locais da função,
15     diferentes das variáveis do procedimento
16     anterior */
17     int x, y;
18     scanf("%d %d", &x, &y);
19     imprimeMaior(x, y);
20     return 0;
21 }
```

Escopo das 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.

Escopo das variáveis

Exemplo

```
1  #include<stdio.h>
2
3  //pode ser acessada a qualquer ponto do programa
4  int contador_global=0;
5
6  void imprimeMaior (int x, int y) {
7      //x e y são visíveis apenas dentro do procedimento
8      if (x > y)
9          printf("%d", x);
10     else
11         printf("%d", y);
12 }
13
14 int main() {
15     //a e b são visíveis apenas na função main
16     int a, b;
17     scanf("%d %d", &a, &b);
18     imprimeMaior(a, b);
19     return 0;
20 }
```

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.

Passagem de parâmetros por referência

- Quando desejamos passar uma variável para uma subrotina de modo que seu valor possa ser alterado pela subrotina, devemos utilizar a Passagem de Parâmetros por Referência.

- O parâmetro da sub-rotina deve ser declarado como **ponteiro** para um tipo

- Na chamada da sub-rotina, deve-se passar o **endereço** da variável (de mesmo tipo do ponteiro) que terá seu valor alterado.

Passagem de parâmetros por referência

```
1  #include<stdio.h>
2
3  void imprimeMaior(int x, int y, int *z) {
4      //int *z ponteiro para o tipo int
5      if (x > y)
6          *z = x;
7      else
8          *z = y;
9  }
10
11 int main() {
12     int a, b, c;
13     scanf("%d %d", &a, &b);
14     //&c endereço da variável de tipo int
15     imprimeMaior(a, b, &c);
16     printf("%d", c);
17     return 0;
18 }
```

Assinaturas de sub-rotinas

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

- Assinaturas correspondem à primeira linha da definição de uma função

- A assinatura de uma sub-rotinas deve aparecer antes do uso desta sub-rotina

- Em geral, colocam-se os protótipos no início do arquivo-fonte, ou em um arquivo separado

Assinaturas de sub-rotinas

```
1   #include <stdio.h>
2
3   void imprimeMaior (int x, int y)
4   {
5       if(x > y)
6           printf("%d\n", x);
7       else
8           printf("%d\n", y);
9   }
10
11  int main( void ) {
12      int x, y;
13      scanf("%d %d", &x, &y);
14      imprimeMaior(x, y);
15      return 0;
16  }
```

Assinatura da sub-rotina

Corpo da sub-rotina

Assinaturas de sub-rotinas

Exemplo

```
1   #include <stdio.h>
2
3   void imprimeMaior (int x, int y);
4
5   int main( void ) {
6       int x, y;
7       scanf("%d %d", &x, &y);
8       imprimeMaior(x, y);
9       return 0;
10  }
11
12  void imprimeMaior (int x, int y) {
13      if(x > y)
14          printf("%d\n", x);
15      else
16          printf("%d\n", y);
17  }
```

Exercício

- Em continuidade ao exercício de cálculo de IMC, escreva uma função com a assinatura

  float calculaIMC (float peso, float altura)

  que retorne o IMC do usuário considerando o peso e altura dados. Faça a leitura dos dados e a impressão do resultado na função main.

Exercício

- Escreva uma função com a assinatura

```
int contaImpar (int n1, int n2)
```

que retorne a quantia de números ímpares que existem no intervalo [n1, n2] (inclusive)

```
n = contaImpar(3, 6);
```

n = 2 (3 e 5)

Exercício

```c
#include <stdio.h>

int contaImpar(int n1, int n2) {
    int menor, maior;
    if(n1 < n2) {
        menor = n1;
        maior = n2;
    } else {
        menor = n2;
        maior = n1;
    }
    int i, n = 0;
    for(i = menor; i <= maior; ++i) {
        if(i % 2 == 1) {
            ++n;
        }
    }
    return n;
}

int main() {
    int n1, n2, n;
    printf("Informe o inicio e o fim do intervalo:\n");
    scanf("%d %d", &n1, &n2);
    n = contaImpar(n1, n2);
    printf("Resultado: %d\n", n);
    return 0;
}
```

Exercício

- Escreva um programa que calcule a potência de um número de acordo com a base e o expoente fornecidos. Separe o cálculo da potência em uma sub-rotina

Exercício

```c
#include <stdio.h>

float potencia(float base, int expoente) {
    float resultado = base;
    int i;
    for(i = 2; i <= expoente; ++i) {
        resultado *= base;
    }
    return resultado;
}

int main() {
    float base, resultado;
    int expoente;
    printf("Informe a base e o expoente:\n");
    scanf("%f %d", &base, &expoente);

    resultado = potencia(base, expoente);

    printf("Resultado: %.2f\n", resultado);
    return 0;
}
```

Exercício

- Escreva uma função com a assinatura

```
int somaIntervalo(int n1, int n2)
```

que retorne a soma dos números inteiros que existem no intervalo [n1, n2] (inclusive). Caso n2 seja menor que n1, a função deve tratar o intervalo como [n2, n1]

```
n = somaIntevalo(3, 6);
```

n = 18 (3 + 4 + 5 + 6)

Exercício

```c
#include <stdio.h>

int somaIntervalo(int n1, int n2) {
    int menor, maior;
    if(n1 < n2) {
        menor = n1;
        maior = n2;
    } else {
        menor = n2;
        maior = n1;
    }
    int i, n = 0;
    for(i = menor; i <= maior; ++i) {
        n += i;
    }
    return n;
}

int main() {
    int n1, n2, n;
    printf("Informe o inicio e o fim do intervalo:\n");
    scanf("%d %d", &n1, &n2);

    n = somaIntervalo(n1, n2);

    printf("Resultado: %d\n", n);
    return 0;
}
```

Exercício

- Escreva um programa que calcule o máximo divisor comum (MDC) de dois números. Separe o cálculo do MDC em uma sub-rotina

Exercício

```c
#include <stdio.h>

int mdc(int n1, int n2) {
    int maior;
    if(n1 < n2) {
        maior = n2;
    } else {
        maior = n1;
    }
    int i, mdc = 0;
    for(i = 1; i <= maior; ++i) {
        if(n1 % i == 0 && n2 % i == 0) {
            mdc = i;
        }
    }
    return mdc;
}

int main() {
    int n1, n2, n;
    printf("Informe dois numeros inteiros:\n");
    scanf("%d %d", &n1, &n2);

    n = mdc(n1, n2);

    printf("Resultado: %d\n", n);
    return 0;
}
```

Bibliografia

- Anido, R., Notas de aula. UNICAMP, disponível em http://www.ic.unicamp.br/~ranido/mc102/, acessado em Maio de 2015.

- Mota, V. F., Notas de aula. UFMG. Disponível em https://sites.google.com/site/virginiaferm/home/disciplinas acessado em Maio de 2015.

- Deitel, P, Deitel, H. C How to Program. 6a Ed. Pearson, 2010.