Departamento de Sistemas de Computação
Universidade de São Paulo
SSC 501 – Introdução a Ciência de Computação I

Aula 9
Conceitos de Função e modularização (continuação)

Seiji Isotani, Rafaela V. Rocha

sisotani@icmc.usp.br

rafaela.vilela@gmail.com

PAE: Armando M. Toda, Geiser Chalco

armando.toda@gmail.com

geiser.gcc@gmail.com

Conceitos de função e modularização – Revisão..

Exemplo:

- Uma **função** é um conjunto de instruções com entradas e saídas bem definidas
  - Nome
  - Entrada(s)
  - Instruções que **executam** a tarefa
  - Instruções que devolvem o resultado (saída(s))

```
FUNCAO nomeDaFuncao(parâmetro1, parâmetro2, ...)
        DECLARE variável1, variáveln
        instrução1
        instruçãon
        .......
        Devolve variáveln;
```

Variáveis locais
Acessíveis somente
dentro da função

Exercício

Crie um programa que consegue adivinhar um número entre 1 a 15 em no máximo 4 tentativas

(assuma que o jogador diga se o número é maior ou menor a cada tentativa incorreta).

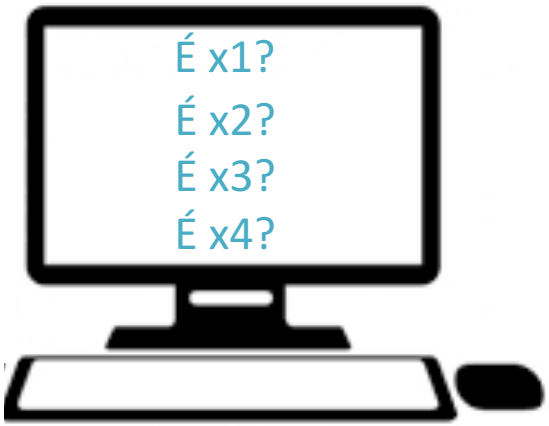
USE APENAS (SE-ENTAO)

Exercício – Análise (1/2)

Crie um programa que consegue adivinhar um número entre 1 a 15 em no máximo 4 tentativas

(assuma que o jogador diga se o **número é maior ou menor a cada tentativa incorreta**).

USE APENAS (SE-ENTAO)



programa

? 1 a 15

jogador

maior

menor

correto

DICA 1 – USE VARIÁVEIS PARA CALCULAR OS VALORES

- EX: Adivinhe o próximo número par:
  - 1 ?
  - 4 ?
  - 99?

```
DECLARE X, PP
LE X
IF (X % 2 == 1 )
    PP = X + 1;
ELSE
    PP = X+ 2;
IMPRIME "Próximo número par é PP?"
```

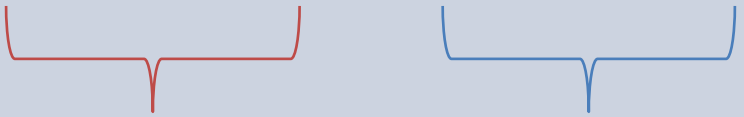
```
DECLARE X, PP
LE X
IF (X == 1)
 PP = 2; // ou IMPRIME "Próximo número par é 2"
IF (X == 2)
 PP = 4; // ou IMPRIME "Próximo número par é 4"
IF (X == 3)
 PP = 4; // ou IMPRIME "Próximo número par é 4"
IF (X == 4)
  PP = 6; // ou IMPRIME "Próximo número par é 6"
IMPRIME "Próximo número par é PP"
          .....
```

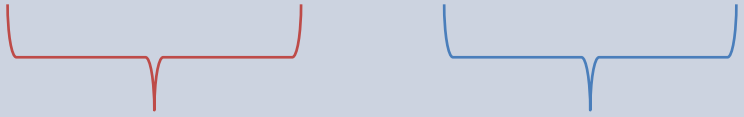
DICA 2 – ENCONTRE PADRÕES

Menor Correto Maior

01 02 03 04 05 06 07 08 09 10 11 12 13 14 15

01 02 03 04 05 06 07 09 10 11 12 13 14 15

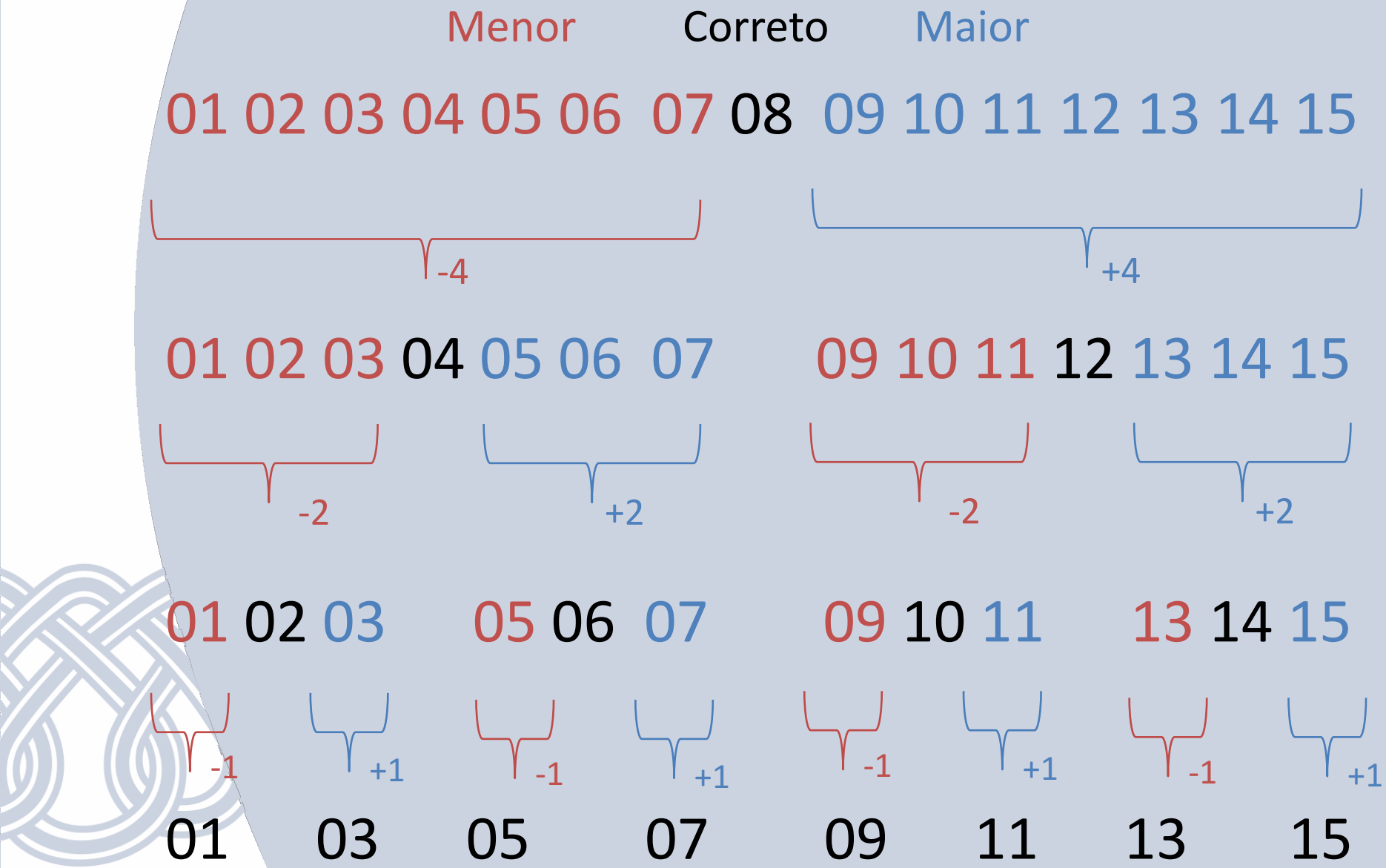




01 02 03 05 06 07 09 10 11 13 14 15

01 03 05 07 09 11 13 15

Exercício – Análise (2/2)



Exercício – Resolução

DECLARE X, R

X = 8

IMPRIME “é x?”

LE R

SE R = “correto” FIM

SE R = “maior” ENTAO x ← x + 4 SENAO x ← x - 4

IMPRIME “é x?”

LE R

SE R = “correto” FIM

SE R = “maior” ENTAO x ← x + 2 SENAO x ← x - 2

IMPRIME “é x?”

LE R

SE R = “correto” FIM

SE R = “maior” ENTAO x ← x + 1 SENAO x ← x - 1

IMPRIME “EU sei que é x !!!!”

```c
int main(void) {
    int x, r;
    x= 8;
    printf("Pense em um número de 1 a 15 \n");
    printf("Tentativa1: É %d? \n", x);
    scanf("%d", &r);
    if (r!=0){
        if (r==1){
            x=x+4;
        }
        else {
            x=x-4;
        }
        printf("Tentativa2: É %d? \n", x);
        if (r!=0){
            if (r==1){
                x=x+2;
            }
            else {
                x=x-2;
            }
            printf("Tentativa3: É %d? \n", x);
            if (r!=0){
                if (r==1){
                    x=x+1;
                }
                else {
                    x=x-1;
                }
                printf("Tentativa4: É %d!!! \n", x);
            }
        }
    }
}
```

Crie uma função chamada chuteCerteiro para diminuir a repetição de código

```
DECLARE x
x ← 8
x ← chuteCerteiro(x, 4)
x ← chuteCerteiro(x, 2)
x ← chuteCerteiro(x, 1)
IMPRIME “EU sei que é x !!!!”

FUNCAO chuteCerteiro(x, y)
        DECLARE R
        IMPRIME “é x?”
        LE R
        SE R = “correto” IMPRIME “ACERTEIIIII ☺” FIM
        SE R = “maior” ENTAO
                DEVOLVE x + y
        SENAO
                DEVOLVE x - y
```

Programa Principal

```
DECLARE x
x ← chuteCerteiro(8, 4)
x ← chuteCerteiro(x, 2)
x ← chuteCerteiro(x, 1)
IMPRIME “EU sei que é x !!!!”
```

Declaração da Função

```
FUNCAO chuteCerteiro(x, y)
        DECLARE R
        IMPRIME “é x?”
        LE R
        SE R = “correto” FIM
        SE R = “maior” ENTAO
                Devolve x + y
        SENAO
                Devolve x – y
```

Exercício 1

- Escreve um algoritmo que calcula a soma de 4 números:

$$n_1^{e1} + n_2^{e2} + n_1^{e3} + n_2^{e4}$$

- Depois verificar se a soma resultante é par ou impar.

Dica: escreva uma função chamada POTENCIA que recebe dois números, a base "b" e o expoente "e" e calcula o valor da potência b^e

Exercício 2

Faça um algoritmo para o cálculo do fatorial (n!) e verifique se este número é ou não primo (apenas dividido por 1 e n).

Crie uma função separada que realize o cálculo do fatorial de um número e uma outra função separada que indique se ele é ou não um número primo.

Programando funções em C

Modularização de Programas

Quebrando programas na Linguagem C - Sub-rotinas / Módulos:

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

char Nome[30];
int  Idade;

main( )
{
  printf("Digite seu nome: ");
  scanf("%s", Nome);

  printf("Digite sua idade: ");
  scanf("%d", &Idade);

  printf("\n");
  printf("Nome: %s\n", Nome);
  printf("Idade: %d\n", Idade);
  printf("\n");

  system("PAUSE");
  return 0;
}
```

Quebrando programas na Linguagem C - Sub-rotinas / Módulos:

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

char Nome[30];
int  Idade;

main( )
{
  printf("Digite seu nome: ");
  scanf("%s", Nome);

  printf("Digite sua idade: ");
  scanf("%d", &Idade);

  printf("\n");
  printf("Nome: %s\n", Nome);
  printf("Idade: %d\n", Idade);
  printf("\n");

  system("PAUSE");
  return 0;
}
```



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

char Nome[30];
int  Idade;

le_nome ( )
{   printf("Digite seu nome: ");
     scanf("%s", Nome);
}

le_idade( )
{   printf("Digite sua idade: ");
    scanf("%d", &Idade);
}

exibe_dados ( )
{   printf("\n");
    printf("Nome: %s\n", Nome);
    printf("Idade: %d\n", Idade);
    printf("\n");
}
```

Modularização de Programas

Quebrando programas na Linguagem C - Sub-rotinas / Módulos:

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

char Nome[30];
int  Idade;

main( )
{
  printf("Digite seu nome: ");
  scanf("%s", Nome);

  printf("Digite sua idade: ");
  scanf("%d", &Idade);

  printf("\n");
  printf("Nome: %s\n", Nome);
  printf("Idade: %d\n", Idade);
  printf("\n");

  system("PAUSE");
  return 0;
}
```



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

char Nome[30];
int  Idade;

le_nome ( )
{   ...  }

le_idade( )
{   ...  }

exibe_dados ( )
{  ...   }

main( )
{
     le_nome();
     le_idade();
     exibe_dados();

    system("PAUSE");
    return 0;
}
```

Programas na Linguagem C - Sub-rotinas / Módulos e Variáveis:

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

char Nome[30];
int  Idade;

le_nome ( )
{   printf("Digite seu nome: ");
    scanf("%s", Nome);
}

le_idade( )
{   printf("Digite sua idade: ");
    scanf("%d", &Idade);
}

exibe_dados ( )
{   printf("\n");
    printf("Nome: %s\n", Nome);
    printf("Idade: %d\n", Idade);
    printf("\n");
}
```

```
main( )
{
      char resp[3];

      resp[0]='s';

      while (resp[0] == 's')
      {
          le_nome();
          le_idade();
          exibe_dados();

          printf("Continuar? (s/n) ");
          scanf ("%s", resp);
       }

}
```

Sub-Rotinas e Variáveis

- Programas podem ter variáveis gerais, as denominadas ***variáveis globais*** que podem ser usadas por qualquer sub-rotina. ***Todos módulos tem acesso as variáveis globais.***

- ***Variáveis globais*** são declarada **FORA** dos blocos, ou seja, fora das sub-rotinas.

- Programas podem ter variáveis proprietárias de um bloco, as denominadas ***variáveis locais*** que podem ser usadas apenas **dentro da sub-rotina** (bloco) onde foram criadas. ***Variáveis locais são acessadas somente dentro do seu bloco.***

Variáveis: escopo

```
#include <stdio.h>
//declaração de variáveis globais

int funcao1() //variáveis locais de parâmetros
{
        // declaração das variáveis locais - função1
        int x;
        return x;
}

main( )
{       int y;
        y = funcao1();
        //declaração das variáveis locais - main()

}
```

Modularização de Programas

Programas na Linguagem C - Sub-rotinas / Módulos e Variáveis:

```
#include <stdio.h>
#include <stdlib.h>
char Nome[30];

le_nome ( )
{   printf("Digite seu nome: ");
    scanf("%s", Nome);
}

le_idade( )
{
    int idade;
    printf("Digite sua idade: ");
    scanf("%d", &idade);
}

exibe_dados ( )
{   printf("\n");
    printf("Nome: %s\n", Nome);
    printf("Idade: %d\n", idade);
    printf("\n");
}
```

Variável Global

```
main( )
{
      char resp[3];

      resp[0]='s';

      while (resp[0] == 's')
      {
          le_nome();
          le_idade();
          exibe_dados();

          printf("Continuar? (s/n) ");
          scanf ("%s", resp);
       }

}
```

Modularização de Programas

Programas na Linguagem C - Sub-rotinas / Módulos e Variáveis:

```
#include <stdio.h>
#include <stdlib.h>
char Nome[30];

le_nome ( )
{   printf("Digite seu nome: ");
    scanf("%s", Nome);
}

le_idade( )
{
    int idade;
    printf("Digite sua idade: ");
    scanf("%d", &idade);
}

exibe_dados ( )
{   printf("\n");
    printf("Nome: %s\n", Nome);
    printf("Idade: %d\n", Idade);
    printf("\n");
}
```

Variável Global

```
main( )
{
    char resp[3];

    resp[0]='s';

    while (resp[0] == 's')
    {
        le_nome();
        le_idade();
        exibe_dados();

        printf("Continuar? (s/n) ");
        scanf ("%s", resp);
     }

}
```

Variável Local do Main()

Programas na Linguagem C - Sub-rotinas / Módulos e Variáveis:

```
#include <stdio.h>
#include <stdlib.h>
char Nome[30];
le_nome ( )
{   printf("Digite seu nome: ");
    scanf("%s", Nome);
}
le_idade( )
{
    int idade;
    printf("Digite sua idade: ");
    scanf("%d", &idade);
}
exibe_dados ( )
{   printf("\n");
    printf("Nome: %s\n", Nome);
    printf("Idade: %d\n", Idade);
    printf("\n");
}
```

Variável Global

Variável Local de Le_Idade()

Cuidado: Idade agora é local

```
main( )
{
    char resp[3];

    resp[0]='s';

    while (resp[0] == 's')
    {
        le_nome();
        le_idade();
        exibe_dados();

        printf("Continuar? (s/n) ");
        scanf ("%s", resp);
    }

}
```

Variável Local do Main()

Modularização de Programas

Funções e Procedimentos da Linguagem C - Sub-rotinas:

Você já usa sub-rotinas a muito tempo!

Sub-Rotinas da Biblioteca Padão do "C" (Libc). Exemplos:

printf **(*"Hello!\n"*);** **=> Executa uma sub-rotina de impressão na tela**

scanf **("%d",valor);** **=> Executa uma sub-rotina de entrada de dados**

strcpy **(*Nome,"Fulano"*);** **=> Copia uma string para uma variável**

num = ***atoi*** **("1234");** **=> Converte uma string em valor inteiro**

preco = ***atof*** **("56.78");** **=> Converte uma string em valor float/double**

x = ***sqrt*** **(nro);** **=> Extrai a raiz quadrada de um número**

x = ***sin*** **(angulo);** **=> Calcula o seno dado um determinado ângulo**

sorteio = ***rand*** **( ) % 10;** **=> Gera um número pseudo-aleatório de 0 a 9**

getchar **( );** **=> Espera que seja pressionada uma tecla**

Modularização de Programas

Sub-rotinas( ) e seus Blocos {} :

```
main ( )          => Main é um bloco que recebeu o nome especial "main"
{
    printf("1");     => Printf é uma sub-rotina executada pelo main
    getchar ( );     => Getchar é uma sub-rotina executada pelo main
    printf("2");     => Sub-rotinas podem ser usadas (chamadas) diversa vezes!
    getchar ( );
}

getchar ( )           => Getchar é um bloco que recebeu um nome
{ ... }                   Sub-Rotina: Getchar  /  Sem Parâmetros
strcpy (Dest, Org)    => Strcpy é um bloco que recebeu um nome
{ ... }                   Sub-Rotina: Strcpy   /  Com Parâmetros
```

Procedimentos (Procedure) e Funções (Function)

- **Funções:** utilizamos quando precisamos calcular algo e retornar um valor
  - Ex: **x = *sin* (angulo); sorteio = *rand* ( ) % 10;**

- **Procedimentos:** quase igual a uma função, mas sem retorno; utilizamos quando precisamos reusar códigos (não precisamos repeti-los)
  - Ex: ***getchar* ( ); *printf* (“Hello!\n”);**

Modularização de Programas

Programas Seqüenciais e Programas Modulares:

```
double calcula_media (N1, N2, N3)
{
   return ( ( N1 + N2 + N3 ) / 3.0 );
}

main ( )
{     double Nota1, Nota2, Nota3;  double Media;

      scanf (“%lf”,&Nota1); scanf (“%lf”,&Nota2); scanf (“%lf”,&Nota3);
      Media = calcula_media (Nota1, Nota2, Nota3);
      prinf (“ Media = %lf \n”, Media);

      scanf (“%lf”,&Nota1); scanf (“%lf”,&Nota2); scanf (“%lf”,&Nota3);
      Media = calcula_media (Nota1, Nota2, Nota3);
      prinf (“ Media = %lf \n”, Media);

}
```

Modularização de Programas

Programas Seqüenciais e Programas Modulares:

```
double calcula_media (N1, N2, N3)
{
   return ( ( N1 + N2 + N3 )
}

faz_media ( )
{
      double Nota1, Nota2, No
      double Media;

      scanf ("%lf",&Nota1);
      scanf ("%lf",&Nota2);
      scanf ("%lf",&Nota3);
      Media = calcula_media (Nota1, Nota2, Nota3);
      prinf (" Media = %lf \n", Media);
}
```

```
main ( )
{
    int cont, total=10;

    for (cont = 0; cont < total; cont ++)
        faz_media ( );
}
```

Modularidade: Procedimentos e Funções

Exercícios

Ex. 1

Considere o programa ao lado e responda as perguntas que seguem...

Faça um "Teste de Mesa" (Simulação da Execução)

Enumere quais são as variáveis deste programa:
Globais
Locais
Parâmetros
E diga se o bloco é uma função ou procedimento

```
1    double calcula_media (int v1, int v2 )
2    {
3         double media;
4         media = ( v1 + v2 ) / 2.0;
5         v1 = v2;
6         return (media);
7    }
8    int main ( )
9   {
10        int  n1, n2;
11        double m1, m2;
12        printf ("Entre 2 números inteiros: ");
13        scanf  ("%d %d",&n1, &n2);
14        m1 = calcula_media ( n1, n2 );
15         m2 = calcula_media ( 10, 7 );
16        return 0;
17    }
```