SSC0101 - ICC1 – Teórica

Introdução à Ciência da Computação I

Revisão de linguagem C

Prof. Vanderlei Bonato: vbonato@icmc.usp.br

Prof. Claudio Fabiano Motta Toledo: claudio@icmc.usp.br

Estrutura Condicional Simples

LINGUAGEM C

```
if (expressão de teste)
    instrução;

if (expressão de teste)
{
    Instrução_1;
    Instrução_2;
    ...
    Instrução_n;
}
```

Exemplo

```
int main()
{
  char ch;
  ch = getche();
  if (ch == 'p')
   {
    printf("\n Tecla p foi pressionada\n");
    }
  system("pause" );
}
```

Estrutura Condicional Composta

LINGUAGEM C

```
if (expressão de teste)
    instrução_1;
else
    instrução_2;
```

```
if (expressão de teste)
{
      instrução_1;
      Instrução_2;
      ...
      Instrução_n;
}
else
{
     instrução_1;
     instrução_2;
      ...
      Instrução_n;
}
```

Comandos aninhados

LINGUAGEM C

```
if (expressão de teste_1)
    if (expressão de teste_2)
        instrução_1;
    else
        instrução_2;
else
    instrução_3;
```

Exemplo de if/if-else aninhados

```
int main()
{
    char ch1, ch2;
    printf("\n  Entre caractere 1:");
    ch1 = getche();
    if (ch1 == 'p')
    {
       printf("\n  Entre caractere 2:");
       ch2 = getche();
       if (ch2 == 'q')
       {
          printf("\n  Você digitou p e q.\n");
       }
       else
       {
           printf("\n  Você digitou p e não q.\n");
       }
    }
    else
    {
       printf("\n  Voce NÃO digitou p e q.\n");
    }
    system("PAUSE");
}
```

Operadores para expressões de teste

LINGUAGEM C

Relacionais

>	maior
>=	maior ou igual
<	menor
<=	menor ou igual
==	igualdade
!=	diferente

Lógicos

&&	E
||	OU
!	Negação (unário)

Exemplo com operadores lógicos

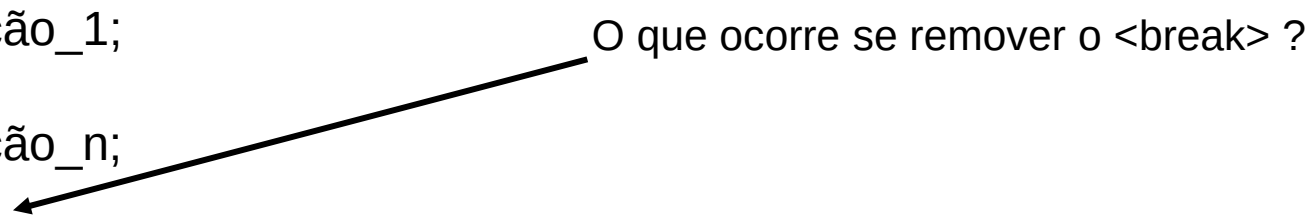
```
int main()
{
    char ch1, ch2;
    printf("\n  Entre caractere 1:");
    ch1 = getche();
    printf("\n  Entre caractere 2:");
    ch2 = getche();

    if (ch1 == 'p' && ch2 == 'q')
    {
       printf("\n  Você digitou p e q.\n");
    }
    else if (ch1 == 'p' || ch2 == 'q')
    {
       printf("\n  Você digitou p ou q.\n");
    }

    if (!(ch1 == 'p') && !(ch2 == 'q'))
    {
       printf("\n  Você NÃO digitou p e nem q.\n");
    }

    system("PAUSE");
}
```

Comando <else if>

```
if (expressão de teste_1)
    instrução_1;
else if (expressão de teste_2)
    instrução_2;
```

Estrutura CASE

```
switch (expressão constante)
{
    case constante_1:
        instrução_1;
        ...
        instrução_n;
        break;
    case constante_2:
        instrução_1;
        ...
        instrução_n;
        break;
    default:
         instrução_1;
        ...
        instrução_n;
}
```



Exemplo com CASE

```
int main()
{
    float num1, num2;
    char op;
    printf("Digite na seguinte ordem: valor 1 \"operador\" valor 2 \n");
    scanf("%f %c %f", &num1, &op, &num2);

    switch(op)
    {
       case '+':
          printf(" = %f\n",num1 + num2);
          break;

       case '-':
          printf(" = %f\n",num1 - num2);
          break;

       default:
          printf("Operador desconhecido \n");
    }

    system("PAUSE");
}
```

Comando for

```
for (inicialização; teste; incremento)
    instrução;

for (inicialização; teste; incremento)
{
    instrução_1;
    instrução_2;
    ...
    instrução_n;
}
```

Comando for

Exemplo:

```
//imprime números de 0 a 9
int main()
{
    int conta;
    for(conta=0; conta<10; conta++)
     //para mais de uma instrução no corpo do for deve-se utilizar chaves
     {
       printf("conta=%d\t",conta);
       printf("conta=%d\n",-(conta-9));
    }
     system("PAUSE");
}
```

Comando for

Exemplos:

```
for(i = 1; i<=10; i++)
   printf("%d ", i);
```
⇒ 1 2 3 4 5 6 7 8 9 10

```
for(i = 1; i<=10; i=i+1)
   printf("%d ", i);
```
⇒ 1 2 3 4 5 6 7 8 9 10

```
for(i = 10; i>=1; i--)
   printf("%d ", i);
```
⇒ 10 9 8 7 6 5 4 3 2 1

Comando for

Exemplos:

```
for(i = 1; i<=10; i=i+2)
   printf("%d ", i);                    ⇒ 1 3 5 7 9

printf("\n\ni=");
for(i = 10; i>=1; i=i-3)
   printf("%d ", i);                    ⇒ 10 7 4 1

printf("\n\ni=");
for(i = -10; i<=10; i=i+5)
   printf("%d ", i);                    ⇒ -10 -5 0 5 10
```

Comando for

Exemplos:

```
//Usando contador float
for(k = 0; k<=1; k += 0.1)
  printf("%3.1f ", k);
```

⇒ 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

```
//Usando contador char
for(letra = 'A'; letra<='Z'; letra++)
  printf("%c ", letra);
```

⇒ A B C ....X Y Z

```
for(letra = 'z'; letra>='a'; letra--)
  putchar(letra);
```

⇒ z y x ....c b a

Comando for - aninhado

- Pode existir n comandos for aninhados

```
for (inicialização1; teste1; incremento1)
        for (inicialização2; teste2; incremento2)
          for (inicialização3; teste3; incremento3)
                    ....
                  for (inicializaçãoN; testeN; incrementoN)
                            instrução;
```

Comando while

```
while (expressão de teste)
   instrução;

while (expressão de teste)
{
   instrução_1;
   instrução_2;
   ...
   instrução_n;
}
```

Comando while

- O “while” pode substituir o “for” do seguinte modo:

```
Inicialização da variável de teste
while(teste)
{
    Incremento da variável de teste;
    ....
}
```

Comando while

- Há uma equivalência entre os comandos while e for.

for (expr1; expr2; expr3) **instrução1;** **Instrução2;**	**expr1;** **while(expr2){** **instrução1;** **expr3;** **}** **Instrução2;**

- Laço for é equivalente ao while, considerando que expr2 ocorre e que não há um comando **continue** no corpo de um laço for.

Comando while

- Exemplo 1

```
int main()
{
    int conta=0;
    int total=0;
    while(conta<10)
    {
       total+=conta;
       printf("conta=%d, total=%d\n", conta, total);
       conta++;
    }
    system("pause");
}
```

Comando while

- Exemplo 2

```
int main()
{
    int conta=0;
    printf("Digite uma frase:\n");
    //13 é o valor do caractere "enter" ou "cr"(carriage return)
    while(getche()!=13)
    {
       conta++;
    }
    printf("A frase possui %d caracteres",conta);

    system("pause");
}
```

Veja a tabela ASCII: http://www.asciitable.com/

Comando while - aninhado

```
while (expressão de teste1)
   while(expressão de teste2)
      while(expressão de teste3)
            ...
             while(expressão de testeN)
                   instrução;
```

Comando do-while

```
do
{
   instrução;
}while(expressão de teste);
```

- Permite executar o bloco mesmo se o teste for falso no início
- Pouco utilizado

Comando do-while

Exemplo 1:

```
//Soma uma série de valores inteiros até receber valor 0
i=0; sum=0;
do{
    sum += i;
    scanf(“%d”, &i);
}while (i>0)
```

Comando do-while

Exemplo 2:

```
//Recebe apenas inteiros positivos
do{
    printf(“Entre com valor inteiro positivo: “);
    scanf(“%d”, &n);
    if(error = (n<=0))
       printf(“\nERROR: Digite novamente!\n\n”);
}while (error);
```

Vetores em C

- **Declaração:**

  <tipo> <nome_variável> [<tamanho>]

- **Limitantes e tamanho:**

  limitante_inferior = 0.

  limitante_superior = tamanho-1.

  tamanho = limitante_superior + 1.

- **Exemplo:**

```
int x[10];              //x[0], x[1], x[2],...,x[9]
float  notas[10];  //notas[0], notas[1], notas[2],...,notas[9]
char   vogais[5];  //vogais[0], vogais[1],....,vogais[5]
```

Vetores em C

- Recomenda-se definir o tamanho de um vetor usando uma constante.
- A constante poderá ser utilizada tanto na declaração do vetor quanto na condição de parada dos laços que percorrem o mesmo.
- Exemplo:

```
#define N 100
int a[N]; // a[0], a[1], ..., a[99]
```

- Normalmente, utiliza-se um laço “for” para processar os elementos de um vetor.
- Exemplo:

```
for(i=0; i<N; ++i)
    sum += a[i];
```

Vetores em C

- Exemplos de inicialização de vetores:

float f[5] = {0.0, 1.0, 2.0, 3.0, 4.0};

  - Quando a lista de valores é menor que o número de elementos, os valores remanescentes são iniciados com valor zero.

float a[100] = {0};

  - Inicia todos os elementos com valor zero.

int a[ ] = {2, 3, 5, -7}; $\Leftrightarrow$ int a[4] = {2, 3, 5, -7};

  - O tamanho do número de elementos do vetor é determinada pela quantidade de valores inicializados.

Vetores em C

- A contagem das posições no vetor começa em 0.
- Incluir mais elementos que o tamanho definido para o vetor é uma fonte de erros.
- Os valores excedentes são atribuídos a uma parte não alocada da memória.
- O espaço alocado em memória é aquele definido quando o vetor foi declarado.

Vetores em C

- Um vetor do tipo “char” pode armazenar “string”
- Note que uma string sempre termina com o caracter *null* (“\0”)
- Exemplo:

Está correto?

```
char nome[4] = "Ana";
char sobrenome[] = {'H','i','t','s'};
printf("%s,%d\n",nome,strlen(nome));
printf("%s,%d\n",sobrenome,strlen(sobrenome));
```

```
char sobrenome[] = {'H','i','t','s','\0'};
```

Exemplo com vetor

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
int main(int argc, char *argv[])
{
  int nota[MAX]; nota[0] = 10;  nota[1] = 20;   nota[2] = 30;
  //int nota[MAX] = {10,20,30};
  //int nota[] = {10,20,30};
  int media,x,acc=0;
  for(x=0; x<MAX;x++){
     acc += nota[x];
     printf("%d\n",nota[x]);
  }
  media = acc/MAX;
  printf("%d\n",media);
  system("PAUSE");
  return 0;
}
```

Note que em C o primeiro elemento do vetor é o índice [0]

Matrizes em C

- Declaração:

  tipo nome_variável[tamanho_dim1] [tamanho_dim2]
  [tamanho_dim3].. [tamanho_dimM];

- Exemplo de declaração de uma matriz:

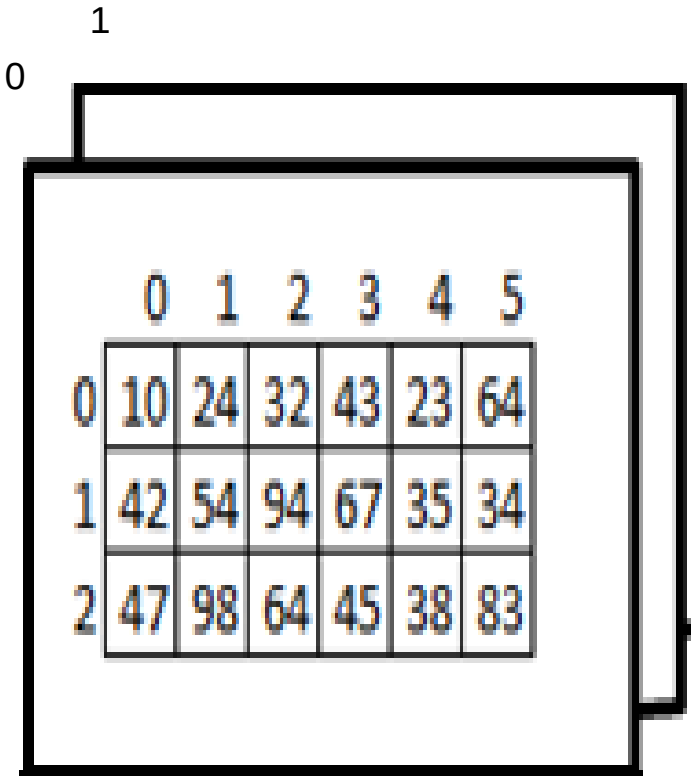
  int tabela [3][6];
  - Tabela com 3 linhas e 6 colunas

  int paginas[3][6][2];
  - Estrutura com 3 linhas, 6 colunas e 2 tabelas de profundidade

Matrizes em C

	0	1	2	3	4	5
0	10	24	32	43	23	64
1	42	54	94	67	35	34
2	47	98	64	45	38	83

tabela[1][2] = 94
tabela[2][1]= 98



pagina [1][2][0] = 94
pagina [2][1][0] = 98

Matrizes em C

- Exemplos de inicializações:

```
int  tabela[3][6] = { {10,24,32,43,23,64},
                      {42,54,94,67,35,34},
                      {47,98,64,45,38,83} };
```

	0	1	2	3	4	5
0	10	24	32	43	23	64
1	42	54	94	67	35	34
2	47	98	64	45	38	83

Dica: pense as inicializações da direita para a esquerda.

```
float mat_A[1][2][3] = {
                         { {5.2,0.9,1.3}, {0.8,4.5,2.3}}
                           };

float mat_B[1][2][3][4] = {
                             {
                    { {5.2,0.9,1.3,4.2}, {0.8,4.5,2.3,6.4},{3.2,3.4,6.3,9.0}},
                    { {8.1,3.4,6.3,7.1}, {2.3,6.1,0.3,9.2},{1.1,3.5,0.1,7.2}}
                              }
                            } ;
```

Matrizes em C

- Exemplos de inicializações (cont.):

  - int a[2][3] = {1,2,3,4,5,6}; ⇔ int a[2][3] = {{1,2,3}, {4,5,6}};
    ⇔ int a[ ][3] = {{1,2,3},{4,5,6}};

  - int a[2][[2][3] = { { {1,1,0}, {2,0,0} },
    { {3,0,0}, {4,4,0} } };
    ⇔ int a[ ][2][3] = { { {1,1}, {2} }, { {3}, {4,4} } };

  - int a[2][2][3] = {0}; //inicia todas as posições com zero

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

void main(){

int ln, cl, pf;

//Iniciando Matriz
float matriz[2][3][4] = {
{ {5.2,0.9,1.3,4.2}, {0.8,4.5,2.3,6.4},{3.2,3.4,6.3,9.0}},
{ {8.1,3.4,6.3,7.1}, {2.3,6.1,0.3,9.2},{1.1,3.5,0.1,7.2}}
};
//Percorrendo Matriz
for(ln=0; ln<2; ln++)
 for(cl=0; cl<3; cl++)
  for(pf=0; pf<4; pf++)
   printf("matriz[%d][%d][%d]=%3.1f\n", ln, cl, pf, matriz[ln][cl][pf]);
}
```

```
matriz[0][0][0]=5.2
matriz[0][0][1]=0.9
matriz[0][0][2]=1.3
matriz[0][0][3]=4.2
matriz[0][1][0]=0.8
matriz[0][1][1]=4.5
matriz[0][1][2]=2.3
matriz[0][1][3]=6.4
matriz[0][2][0]=3.2
matriz[0][2][1]=3.4
matriz[0][2][2]=6.3
matriz[0][2][3]=9.0
matriz[1][0][0]=8.1
matriz[1][0][1]=3.4
matriz[1][0][2]=6.3
matriz[1][0][3]=7.1
matriz[1][1][0]=2.3
matriz[1][1][1]=6.1
matriz[1][1][2]=0.3
matriz[1][1][3]=9.2
matriz[1][2][0]=1.1
matriz[1][2][1]=3.5
matriz[1][2][2]=0.1
matriz[1][2][3]=7.2
```

Matrizes em C

- Numa declaração do tipo

  int a[7][9][2]

  o compilador irá alocar espaço para 7x9x2 valores inteiros contíguos.

- O mapeamento desses valores faz com que

  a[i][j][k] $\Leftrightarrow$ *(&a[0][0][0]+9*2*i+2*j + k)

Funções em linguagem C

<tipo_retornado> <nome_função>(<lista_dos_parametos>)

```
{
        <declarações>
        <instruções>
}
```

Exemplo:

```
int fatorial (int n)        /* cabeçalho da função*/
{                             /* início do corpo da função*/
 int i, product = 1;
for (i=2; i<=n; ++i)
  product *= i;
return product;
}
```

Return

- As funções retornam um resultado que deve ser do mesmo tipo para o qual a função foi declarada.

<**tipo_retornado**> <nome_função>(<lista_dos_parametos>)

int fatorial (int n)

- O comando **return** é responsável por encerrar a execução da função e retornar o valor daquele tipo.

return product;

Return

- Se um tipo não é especificado para uma função, o tipo **int** será o *default*.

```
int all_add( int a, int b)
{
 int c;
 ....
return  (a+b+c);
}
```

⇔

```
 all_add( int a, int b)
{
 int c;
 ....
return  (a+b+c);
}
```

Return

- O valor retornado é convertido, se necessário, para o tipo retornando pela função

```
 float add( int a, int b)
{
 int soma;
  soma  = a+b;
return  soma;
}
```

Return

- Recomenda-se limitar a função para que tenha um único **return** visando facilitar a compreensão da função.

- Todavia, o uso de mais que um **return** também pode tornar o código mais legível.

- Desta forma, a quantidade de **return** em uma função deve facilitar o entendimento e a manutenção do código.

Return

- Exemplo:

```
double absolute_value(double x)
{
  if(x>-0.0)
    return x;
  else
    return –x;
}
```

Void

- As sub-rotinas na linguagem C podem ser encaradas todas como funções.

- A palavra reservada **void** na declaração de uma sub-rotina indica que se trata de uma função que não retorna valor.

- O uso de **void** no lugar de uma lista de parâmetros indica que a função não utiliza argumentos (lista de parâmetros).

Void

- Exemplos:

```
void nadafaz(void) { }

void wrt_endereço(void){
 printf("%s\n%s\n%s\n%s\n%s\n\n",
    "                  **************************",
    "                   ***   SANTA CLAUS  *",
    "                    ***   NORTH POLE   *",
    "                    ***   EARTH        *",
    "                  **************************");
}
```

Void

- Os trechos abaixo são equivalentes

  void func() $\Leftrightarrow$ void f(void)

- A declaração abaixo, considerando a linguagem C tradicional, significa que o número de argumentos da função não é conhecido.

  int func();

- Isso ocorre pelo fato de **void** não ser uma palavra reservada na linguagem C tradicional.

Declarações de funções

- Exemplos:

```
float func(x,  y) /*  C  tradicional */
int x
float  y;
{...
<corpo_da_função>
...}

float func(int x,  float y) /* ANSI C  tradicional */

{...
<corpo_da_função>
...}
```

Declarações de funções

- Uma protótipo da função indica ao compilador o número e o tipo de argumentos que devem ser passados para a função e o tipo de valor que deve ser retornado pela função.

<**tipo_retornado**> <nome_função>(<lista_dos_parametos>);

- A lista dos parâmetros apresenta os tipos separados por vírgula, onde os identificadores são opcionais.

float func(int, float); ⇔ float func(int x, float y);

Declarações de funções

- Se uma função, por exemplo, func(x) é chamada antes de sua declaração, definição ou protótipo, o compilador assume a declaração abaixo como default

```
int func();
```

- A maioria dos compiladores precisa conhecer os tipos de retorno e os parâmetros, antes que o programa principal faça uma chamada à sub-rotina.

Declaração de funções

- Exemplo1:

```
#include <stdio.h>
#define N 7
long   power(int, int);
void   prn_heading(void);
void   prn_tbl_of_powers(int);

int main(void)
{
 prn_heading();
Prn_tbl_of_powers(N);
return 0;
}
```

Declarações de funções

```
void prn_heading(void)
{
   printf(“\n::: A TABLE OF POWERS :::\n\n”);
}

void prn_tbl_of_powers(int n)
{
       int i,j;
      for(i=1; i<=n; ++i){
       for(j=1; j<=n; ++j)
           if(i==j)
              printf(“%ld”, power(i,j));
           else
               printf(“%9ld”,power(i,j));
       putchar(‘\n’);
      }
}
```

```
void power(int m, int n)
{
       int i;
       long product = 1;

      for(i=1; i<=n; ++i){
         product *=m;

      return product;
}
```

Declarações de funções

- Exemplo2:

```
#include <stdio.h>
#define N 7
void prn_heading(void)
{....}
long power(int m, int n)
{....}
long prn_tbl_of_powers(int n)
{...printf("%ld", power(i,j));...}

int main(void)
{
      prn_heading();
      prn_tbl_of_powers(N);
      return 0;
}
```

Declarações de funções

- Exemplos 1 e 2: Saída

::: A TABLE OF POWERS ::::

1	1	1	1	1	1	1
2	4	8	16	32	64	128
3	9	27	81	243	729	2187

.....

Passagem de parâmetros por valor

- Uma cópia dos valores das variáveis é passada à função.
- As modificações executadas nos valores fornecidos estão restritas ao escopo da função.
- O valor original da variável não é alterado.

Exemplo:

```
#include<stdio.h>
int comput_sum(int n);
int main(void) {
      int n=3, sum;
      printf(“%d\n”,n);       /* 3 é exibido*/
      sum = compute_sum(n);
      printf(“%d\n”,n);       /* 3 é exibido */
      printf(“%d\n”,sum); /* 6 é exibido*/
      return 0;
}

int compute_sum(int n){
       int sum = 0;
       for(; n > 0; --n)
            sum+= n;
      return sum;
}
```

Passagem de parâmetros por referência

- O endereço de memória da variável é fornecido à função e não uma cópia do valor da variável.
- Qualquer alteração executada pela função ocorre na posição de memória fornecida.
- Por isso, as alterações permanecem quando a função é encerrada.
- Ponteiros, que serão apresentados em breve, são utilizados nas passagens por referência.

- Exemplo:

```
#include <stdio.h>
void swap(int *, int *);
int main(void){
 int i=3, j=5;
swap(&i, &j);
printf("%d   %d\n", i, j);  /* 5  e 3 são exibidos */
return 0;
}
void swap (int *p, int *q){
   int tmp;
   tmp = *p;
   *p = *q;
    *q = tmp;
}
```

Passagem de parâmetros por referência

- As variáveis i e j são passadas por referência, ou seja, o endereço de memória das variáveis é repassado à função.

swap (&i, &j)

- Os ponteiros *p e *q, declarados no argumento na função swap, passam a referenciar a posição de memória das variáveis i e j.

void swap (int *p, int *q)

```c
#include<stdio.h>
float valorHora = 30;     //VARIÁVEIS GLOBAIS
int totalHorasMes = 160;

float calculaSalario(int, float*, float); //Protótipos

void main(){ //Programa Principal
float salario, salarioReferencia = 1000, gratificacao = 500;
int totalHoraTrabalhada;
char nome[30];

printf("Nome do fucionario:"); gets(nome);
printf("Total de horas trabalhada:"); scanf("%d",&totalHoraTrabalhada);

salario = calculaSalario(totalHoraTrabalhada,&gratificacao, salarioReferencia);
printf("Valor salario: R$%5.2f\n", salario);
printf("*** Valor original da gratificacao FOI alterado: R$%5.2f\n", gratificacao);
}

float calculaSalario(int totalHora, float *gratificacao, float salarioRef){

    float salario; //VARIÁVEIS LOCAIS
    int dif;

    //Ajusta a gratificacao
    dif = totalHora - totalHorasMes;
    if(dif>=0)
        *gratificacao = 1000;
    else if(dif<0)
        *gratificacao = 0;

    printf("*** Valor da gratificacao foi alterado na funcao: R$%5.2f\n", *gratificacao);
    salario = totalHora*valorHora + *gratificacao;

    return salario; //RETORNA O VALOR FINAL DO SALÁRIO (TIPO float)
}
```

```
Nome do fucionario:Maria Silva
Total de horas trabalhada:180
*** Valor da gratificacao foi alterado na funcao: R$1000.00
Valor salario: R$6400.00
*** Valor original da gratificacao FOI alterado: R$1000.00
```

Passagem de parâmetros por referência

Passagem por Valor

Main	valor
totalHoraTrabalhada	180
gratificacao	500
salarioReferencia	1000
salario	

calculaSalario	valor
totalHora	180
gratificacao	~~500~~ 1000
salarioRef	1000
return salario	6400

Passagem por Referência

Main	valor
totalHoraTrabalhada	180
gratificacao	~~500~~ 1000
salarioReferencia	1000
salario	

calculaSalario	valor
totalHora	180
gratificacao	~~500~~ 1000
salarioRef	1000
return salario	6400

Passagem de parâmetros por referência

- Vetores e matrizes
  - São passados sempre como referência.
  - Os colchetes, após o nome do vetor passado a uma função, indicam que o parâmetro é um vetor. Não precisa especificar o tamanho do vetor.

```
int soma_vetor(int vetor[], int elementos)
```

  - Exemplo: Formas possíveis de se declarar a função func() para receber o vetor int vet[100]

```
func(int x[100]);
func (int x[]);
func(int *x);
```

Passagem de parâmetros por referência

- Vetores e matrizes
  - Ao passar uma matriz bidimensional, se for preciso acessar entradas específicas, o número de colunas precisa ser fornecido.

```
int soma_matriz(int matriz[][5], int linhas)
```

  - Se não há necessidade de acessar entradas específicas da matriz, ela poderá ser tratada como um vetor. Nesse caso, o número de elementos da matriz deverá ser fornecido.

```c
#include<stdio.h>
#define LN  2
#define CL  5

int soma_vetor(int[], int);
int soma_matriz(int , int colunas,int matriz[][colunas]);

void main(){
    int vetor[CL]={10, 20, 30, 40, 50};

    int bidim[LN][CL]={{10,20,30,40,50},
                       {60,70,80,90,100} };
    printf("Soma dos valores do vetor:%d\n", soma_vetor(vetor,CL));
    printf("Soma dos valores da matriz:%d\n\n", soma_matriz(LN,CL,bidim));
    printf("Soma dos valores do matriz:%d\n", soma_vetor(bidim,LN*CL));
}

int soma_vetor(int vetor[], int elementos){
    int i,soma;

    soma=0;
    for(i=0; i<elementos; i++)
     soma += vetor[i];

    return soma;
}

int soma_matriz(int linhas, int colunas,int matriz[][colunas]){
    int i,j,soma;

    soma=0;
    for(i=0; i<linhas; i++)
     for(j=0; j<colunas; j++)
       soma += matriz[i][j];

     return soma;
}
```

```
Soma dos valores do vetor:150
Soma dos valores da matriz:550

Soma dos valores do matriz:550
```

Recursão

- Uma função é recursiva quando chama a si própria.
- Exemplo:

```
int sum(int n){
 if (n<=1)
   return n;
 else
   return (n+sum(n-1));
}
```

Entrada	Saída	
Sum(1)	1	
Sum(2)	2 + sum(1)	2+ 1
Sum(3)	3 + sum(2)	3 + 2 + 1
Sum(4)	4 + sum(3)	4 + 3 + 2 + 1

Recursão

- Exemplo:

```
int fat(int n){
 if (n<=1)
   return 1;
 else
   return (n*fat(n-1));
}
```

- A partir de determinado valor de n, as saídas do programa podem fornecer valores errados.
- Qual o motivo?

Recursão

- Exemplo: Sequência de Fibonacci.

f(0)=0, f(1)=1, f(i+1)=f(i)+f(i-1), i=1,2,...

```
int fib(int n){
 if(n<=1)
  return n;
 else return((fib(n-1)+f(n-2));
}
```

n	F(n)	Número de chamadas da função
0	1	1
1	1	1
2	1	3
...	...	...
23	28657	92735
24	46368	150049

Referências

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

VICTORINE VIVIANE MIZRAHI, Treinamento em Linguagem C – Módulo 1 e Módulo 2, Makron Books, 1990.

FIM Aula 15