RESUMÃO LINGUAGEM C

Autor: Everton M. Messias | Última atualização: 08/03/2016

C é uma **linguagem** de programação compilada criada em 1972 por Dennis Ritchie no AT&T Bell Labs para desenvolver o sistema operacional Unix.

```
#include <stdio.h> /* Biblioteca de entrada e saída (teclado/vídeo) */
#include <stdlib.h> /* Biblioteca para usar a função system() */
main () { /* Função Principal, sem retorno - VOID */
printf ("Hello world!\n\n"); /* Função printf() , escrita , \n pula linha */
system ("pause"); /* Função system() - executa um comando do SO */}
```

Mapa de Memória: Um programa C compilado cria e usa quatro regiões de memória: O Código propriamente dito, as Variáveis Globais, a Pilha (variáveis locais e end. de funções) e o Heap (região de memória livre usada pelo programa para alocação dinâmica, listas e árvores).

Constantes : Valor fixo que não se modifica durante a execução. Ex: #define PI 3.14159

Variáveis : Um ou mais caracteres - números, letras (maiúsculas ou minúsculas) – É uma posição nomeada de memória que representa um valor. Se estiver fora de qualquer função é variável global.

```
#include <stdio.h>
#include <conio.h> /* Biblioteca para usar a função getch() */
int main (){
char letra; // para obter o cod ASCII troque char por int ...
letra=getch();  // No Unix usar: scanf("%c", & letra). Desvantagem; tem q pressionar <ENTER>
printf ("Voce pressionou a tecla %c",letra);}
```

Tipos de Variáveis:

Tipo	O que é	Tamanho em bits	Formato I/O	Faixa de Abrangência
char	caracteres	8	%c	2^8 ; -128 até 128
int	números inteiros	16 short, 32 long	%d	2^{16} ; -32768 até 32768
float	números reais	32	%f	3,4 x 10^{-38} até 3,4 x 10^{38}
double	números reais	64	%lf	$(+/-)10^{-308}$ $(+/-)10^{308}$
void	vazio	0	0	0

Obs: %s – string , %e – notação científica.

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   char letra = 'a';
   int num = 10;
   float real = 5.25;
   printf ("%c\n",letra);
   printf ("%d\n",letra);
   printf ("%d\n",num);
   printf ("%.2f\n",real);
   system ("pause");}
```

Operadores

+ (adição)	== (igual)	&& (e)	= (atribui)
- (subtração)	!= (diferente)	|| (ou)	++ (incremento)
* (multiplicação)	> (maior)	! (não)	-- (decremento
/ (divisão)	< (menor)		
% (resto)	>= (maior-igual)		
	<= (menor-igual)		

Bibliotecas de Cabeçalho, alguns ex:

stdio.h – funções de entrada e saída de dados
stdlib.h – funções de uso genérico
string.h – funcões de tratamento de string
math.h – funções matemáticas
conio.h – funçõe específicas

Operador ? : Exp1 ? Exp2 : Exp3; ==> V:F
Ex : x=10;
y = x>9 ? 100 : 200 // y recebe 100 pois x>9 é verdadeiro
(x>9) ? printf ("verdadeiro") : printf ("falso");

Condição:

```
if (condição) {
                }
else {
     }

switch (opcao) {                    // executa um bloco conforme a opcao
          case 1:
                    xxx
                    break;  // quebra loop
          case 2:
                    xxx
                    break;
          default:
                    xxx
          }
```

Repetição:

```
while (condição) {                  // testa e depois faz o loop..
                 }


do {                // primeiro faz um loop e depois testa..
   }
while (condição);

for (cont=0; cont < 20; cont ++) {                  // ( início ; condição ; incremento )
                                        }
goto – use apenas em último caso … Ex.: … REFAZ: xxx; goto REFAZ;
```

Vetores: São variáveis compostas homogêneas unidimensionais (seu nome é seu end).

int notas[5]; // cinco elementos inteiros (contados de 0 até 4)

int vetor[5] = {0,1, 2, 3, 4};

for (i=0; i<5; i++){ printf ("Digite um valor:"); scanf ("%d", &valores[i]); }

Matrizes: São variáveis compostas homogêneas bidimensionais (Linha x Coluna).

int matriz[2] [3] = {0,1,2,3,4,5} == {{0,1,2} , {3,4,5}} ; // numeros das colunas

```
# include <stdio.h>
main()
{
   int L,C,matriz[2][3];
      for (L=0;L<2;L++){
         for (C=0;C<3;C++){
         printf ("Digite um valor:");
         scanf ("%d",& matriz[L][C]);}}

   for (L=0;L<2;L++){
         for (C=0;C<3;C++){
         printf ("%d ", matriz[L][C]);}printf ("\n");}
}
```

Strings: Não existe na linguagem C um tipo de dado string . Utiliza-se um vetor de caracteres (tipo char) que termina com um caractere de controle '\0' (colocado pelo compilador).

Ex: char nome[5]={ 'a' , 'l' , 'u' , 'n' , 'o' }; char nome[6] = "aluno";

```
# include <stdio.h>
#include <stdlib.h>
#include <string.h>
main(){
   char string[100];
   printf ("\nDigite um nome completo: ");
   scanf ("%s", string); // não precisa do '&' pois o nome do vetor é um ponteiro !
   printf ("\nImprimindo com scanf ==> %s ",string);
   __fpurge(stdin);      // LIMPAR O BUFFER; para windows use: fflush(stdin);
   printf ("\n\nDigite outro nome completo: ");
   fgets (string,100,stdin);      // simplesmente: gets (string); para windows
   printf ("\nImprimindo com fgets ==> %s",string); }
```

Obs.: scanf – não admite espaços ; gets e fgets admitem espaços e fazem a contagem até \0

Biblioteca string.h, principais funções:

strcpy (destino,origem) – copia
strcat (destino,origem) – anexa no fim da destino
strlen (palavra) – tamanho da string
strcmp (palavra1,palavra2) – compara , se forem iguais retorna 0

Struct : São variáveis compostas heterogêneas (int, char, float ou outra struct).

```
struct aluno {
          char nome [30];
          int idade;
       };
struct aluno dados; //ou diretamente: struct aluno dados = {“Fulano”,38};
```

OBS: So pode atribuir dados1 = dados se for estrutura igual, isto é : struct aluno

Ex. struct com array ou vetor de struct

```
# include <stdio.h>
# include <stdlib.h>
# define MAX 2 // vetor com MAX campos

struct aluno {  // estrutura tipo aluno com campos nome e idade
       char nome [30];
       int idade;};

main() {
   struct aluno dado[MAX];    // declarando variável dado[MAX] do TIPO struct aluno
   int i;
   for (i=0;i<MAX;i++){
   fflush(stdin);
   printf ("\nDigite o nome do aluno %d : ", i+1);
   gets (dado[i].nome);
   printf ("\nDigite a idade do aluno %d : ", i+1);
   scanf ("%d", & dado[i].idade); }
   printf ("\n\nLISTA DE ALUNOS\n\n");
   for (i=0;i<MAX;i++){
   printf ("%s : %d anos\n", dado[i].nome,dado[i].idade); }
   printf ("\n");
   system ("pause");
   return 0; }
```

typedef: Define “apelidos” aos tipos (struct, int, char, float, etc). Ex

```
typedef int inteiro;
typedef struct aluno gente

int idade => inteiro idade;
struct aluno dados => gente dados (não pecisa mais escrever “struct aluno”)
```

Ponteiros : São variáveis cujo conteúdo é um endereço de memória.

Ex:

```
#include <stdio.h>
int main() {
int x=10,*p=NULL;
p = &x;
printf ("x = %d\n",x); // 10, o valor em x
printf ("*p = %d\n",*p); // 10 , aponta para o end de x que contem 10
printf ("&x = %d\n",&x); // 2293532, o end de x
printf ("p = %d\n",p); // 2293532, o end de x pois ele recebeu o end de x
printf ("&p = %d\n",&p); } // 2293528 o end do ponteiro ==> 4 bytes menos pois é inteiro
```

Funções: tipo nome (parametros) { conj de declarações e comandos }

- Pode ter passagem por valor ou referência (ponteiros).
- Pode-se declarar antes (utilizando protótipos) ou depois do main().
- Pode-se passar parametros ou não.
- Recursão: Uma funcão dentro da outra.
- Modularização : Arquivos de cabeçalho. Ex. # include “equacao2g.h”

Ex1: Passagem por valor em uma função recursiva. (ex. Cálculo do FATORIAL)

```
# include <stdio.h>
double fatorial (int n) {
   if (n != 0)return n*fatorial(n-1);
   else return 1 ;}
main(){
   int numero;
   double resposta;
   printf ("\nDigite um numero: ");
   scanf ("%d", & numero);
   resposta=fatorial(numero);
   printf ("\nA resposta eh: %.0lf \n\n",resposta); // para double %f ou %e
   return 0;}
```

Ex2: Passagem por referência de um vetor. (ex. Cálculo da EQ 2º GRAU)

```
#include <stdio.h>
float equacao2g (float *e, int t) { //recebe o end num ponteiro e o tamanho do vetor
            float delta;
            if (e[0] == 0){printf("\n\nERRO: 'a' nao pode ser zero!\n\n");exit(0);}
                                    delta=(e[1]*e[1])-(4*(e[0])*(e[2]));
                                    if (delta >= 0){
                                    e[3]=(((-1)*(e[1]))+(sqrt(delta)))/(2*(e[0]));
                                    e[4]=(((-1)*(e[1]))-(sqrt(delta)))/(2*(e[0]));
                                    return *e;} //retorna o que aponta, isto é: os dados
                                    else {printf ("\n\nERRO: nao existe raiz real\n\n");exit(0);}}

main () {
   float dados[5];
   printf ("\nDigite o valor do termo a : ");
   scanf ("%f", &dados[0]);
   printf ("\nDigite o valor do termo b : ");
   scanf ("%f", &dados[1]);
   printf ("\nDigite o valor do termo c : ");
   scanf ("%f", &dados[2]);
   *dados=equacao2g (dados,5); //envia s/ '&' ( e o tam vetor), recebe em: dados[5] ou *dados
   printf ("\n\nx1 = %.2f e x2 = %.2f\n\n",*(dados+3),dados[4]);}
```

Ex3: Passagem de struct (ex. Cálculo da EQ 2º GRAU)

```
#include <stdio.h>
typedef struct {
      float a,b,c,x1,x2;
}eq2g;

eq2g equacao2g (eq2g *e) { //recebe o end dos dados num ponteiro
            float delta;
            if (e->a == 0){printf("\n\nERRO: 'a' nao pode ser zero!\n\n");exit(0);}
                                    delta=(e->b*e->b)-(4*(e->a)*(e->c));
                                    if (delta >= 0){
                                    e->x1=(((-1)*(e->b))+(sqrt(delta)))/(2*(e->a));
                                    e->x2=(((-1)*(e->b))-(sqrt(delta)))/(2*(e->a));
                                    return *e;} //retorna o que aponta, isto é: os dados
                                    else {printf ("\n\nERRO: nao existe raiz real\n\n");exit(0);}}
main () {
   eq2g dados; //duas variaveis do tipo eq2g
   printf ("\nDigite o valor do termo a : ");
   scanf ("%f", &dados.a);
   printf ("\nDigite o valor do termo b : ");
   scanf ("%f", &dados.b);
   printf ("\nDigite o valor do termo c : ");
   scanf ("%f", &dados.c);
   dados=equacao2g (&dados); // envia o end e recebe dados
   printf ("\n\nx1 = %.2f e x2 = %.2f\n\n",dados.x1,dados.x2);}
```

Alocação Dinâmica: Antes associamos o nome do vetor a um ponteiro, agora associaremos um ponteiro a um vetor de tamanho indefinido

SIZEOF – retorna o tamanho de um tipo.

```
# include <stdio.h>
struct exemplo {
   int x,y,z,w;};
main() {
printf ("char = %d\n",sizeof (char)); // 1 byte
printf ("int = %d\n",sizeof (int)); // 4 bytes
printf ("float = %d\n",sizeof (float)); // 4 bytes
printf ("double = %d\n",sizeof (double)); // 8 bytes
printf ("struct = %d\n",sizeof (struct exemplo));} // 4*4 = 16 bytes
```

MALLOC – Aloca memória em forma de vetor de um determinado tipo.

```
# include <stdio.h>
# define T 5 // esse valor pode ser pedido em scanf
main() {
int *p , i; // (int*) conversão de tipo
p = (int*) malloc(T * sizeof(int)); // criando um vetor de T posições int
if (p==NULL){printf ("ERRO");exit(1);} // se der erro sai fora
for (i=0;i<T;i++){printf ("Digite o num %d : ", i+1);scanf ("%d", &p[i]);} // tratando o vetor
for (i=0;i<T;i++){printf ("%d ", p[i]);} // imprimindo
free(p);} // liberando memoriano só no final
```

CALLOC – semelhante ao MALLOC, identifica quantidade e tamanho preenchendo com zeros;

```
# include <stdio.h>
main() {
int n, i, *p , x = sizeof(int);
printf("Qtos numeros INT deseja armazenar ?\n");
scanf ("%d", &n);
p = calloc(n,x); // criando um vetor de zeros de N posições de tamanho X Bytes
printf("\nPonteiro antes:\n");
for (i=0;i<n;i++){printf ("%d, ", p[i]);}
printf("\n\nInserindo Dados...\n");
for (i=0;i<n;i++){printf ("Digite o %do num: ", i+1); scanf ("%d", &p[i]);} // tratando o vetor
printf("\nPonteiro Agora:\n");
for (i=0;i<n;i++){printf ("%d, ", p[i]);}
free(p);} // liberando memoriano final
```

Arquivos: Para trabalhar com arquivos, C precisa de um ponteiro específico. Ex.:

```
FILE *ponteiro;
ponteiro = fopen (“nome_do_arquivo”, “modo”);
```

Modos mais usados:

- r – leitura
- w – escrita
- at – escrita no fim do arquivo de texto
- rb – leitura binária
- wb – escrita binária

```
#include <stdio.h>
#include <ctype.h>
// Converte textos de minusculo para MAIUSCULO
main(){
        FILE *f1, *f2;
        f1 = fopen ("minusculo.txt","r");
        f2 = fopen ("maiusculo.txt","at");
        if (f1 == NULL || f2 == NULL){
        printf("ERRO");exit(0);}
        char c = fgetc (f1);
        while (!feof(f1)){
        fputc (toupper(c),f2);
        c = fgetc (f1);
        }fclose(f1);fclose(f2);  }
```

```
#include <stdio.h>
main(){
	FILE *arq;
	int op;
	printf ("Digite (e)screver ou (l)er\n");
	op = getc (stdin);
	switch (op){
	case 108:
	arq = fopen ("teste.txt","r");
	if (arq == NULL){printf("ERRO");exit(0);}
	char txt[20];
	fgets (txt,20,arq);
	printf("%s",txt);
	break;
	case 101:
	arq = fopen ("teste.txt","at");
	if (arq == NULL){printf("ERRO");exit(0);}
	char texto[20]="\nMeu programa em C";
	fputs (texto,arq);
	break;
	default:printf ("Opcao Errada");break;
	fclose(arq);}}

#include <stdio.h>
main(){
	FILE *arq; // ponteiro tipo FILE
	int op;
	printf ("LISTA DE NOMES\n");
	printf ("Digite (e)screver ou (l)er\n");
	op = getc (stdin); // recebe opção do teclado
	switch (op){
	case 101:// e em ascii
	arq = fopen ("teste.txt","at");
	if (arq == NULL){printf("ERRO");exit(0);}
	char nome[30];
	printf("Digite seu nome: \n");
	__fpurge(stdin); // LIMPAR O BUFFER; para windows use: fflush(stdin);
	fgets(nome,30,stdin);
	printf("\nGravado: %s\n",nome);
	fprintf(arq,"%s",nome);
	break;
	case 108: // l em ascii
	arq = fopen ("teste.txt","r"); // abre arquivo para leitura
	if (arq == NULL){printf("ERRO");exit(0);}
	int pos;
	printf ("Qual posicao deseja ler ?");
	scanf ("%d",&pos);
	fseek (arq,pos*sizeof(char),SEEK_SET);
	char txt[20];
	fgets(txt,20,arq); // fscanf (arq,"%s",txt);
	printf("%s",txt);
	break;
	default:printf ("Opcao Errada");break;
	fclose(arq);}}
```