

1. A LINGUAGEM C

“C é uma linguagem de programação de finalidade geral que permite economia de expressão, modernos fluxos de controle e estruturas de dados e um rico conjunto de operadores.” Kernighan & Ritchie 1978

1.1 Estrutura Geral de um Programa em C

Diretivas de Pré-processamento

```
#include ....
#define ....
```

Declarações Globais

```
Protótipos de Funções /*tipo de retorno e tipo dos parâmetros */
```

```
main( ) /* função principal – marca o início da execução
do programa*/
```

```
{
  declarações locais;
  comandos;
  ....
}
```

```
Tipo função1 (declaração de parâmetros)
```

```
{
  declarações locais;
  comandos;
}
```

.....

```
Tipo funçãoN (declaração de parâmetros)
```

```
{
  declarações locais;
  comandos;
}
```

1.2. Linguagem de Pseudo-Código (Algoritmo) X Linguagem C

início {algoritmo que calcula o perímetro e a área de uma circunferência de raio R (fornecido pelo usuário)}

inteiro: R;

real: Perm, Area, PI;

PI ← 3.14159;

imprima(“Entre com o valor do raio:”);

leia(R);

Perm ← 2 * PI * R;

Area ← PI*R**2;

imprima(“O perímetro da circunferência de raio”,R,”é”,Perm);

imprima(“e a área é “,Area);

fim



/* programa que calcula o perímetro e a área de uma circunferência de raio R (fornecido pelo usuário)*/

#include <stdio.h> /* inclui diretivas de entrada-saída*/

#include <math.h> /*inclui diretivas das funções matemáticas*/

void main(void)

{

int R;

float Perm, Area, PI;

PI = 3.14159;

printf(“Entre com o valor do raio:”);

scanf(“ %d”,&R);

Perm = 2 * PI * R;

Área = PI* pow(R,2);

printf(“O perímetro da circunferência de raio %d é %.2f \n”,R,Perm);

printf(“e a área é %.2f“,Area);

}

1.3 Tipos Básicos de Dados em C

char → armazena caracteres ou números literais Ex: 'a' '1' '\n'

int → armazena números inteiros Ex. 50, 017

float → armazena números com ponto flutuante em precisão simples (até 6 ou 7 dígitos significativos dependendo da máquina)

Ex: 6.5 -12.4 1.2e-3 -0.00013

double → armazena números com ponto flutuante em precisão dupla (até 15 ou 16 dígitos significativos dependendo da máquina)

Ex: 0.51231298967 -1.3e-15

1.4. Modificadores

unsigned → armazena número sem sinal (positivo) Ex unsigned int

short → reduz os limites de variação Ex . short int

long → amplia os limites de variação Ex. long int

void → tipo usado em funções para indicar parâmetros ou valores de retorno vazio
Ex int função-soma(void)

Tabela de variáveis, intervalos de variação e total alocado na memória

Tipo de dados	Variação	Total de Bytes Utilizados
char	0 a 255	1
int	-32768 a 32767	2
short int	-128 a 127	1
unsigned int	0 a 65535	2
long int	-4294967296 a 4294967296	4
float	Aprox. 6 casas decimais	4
double	Aprox. 12 casas de precisão	8
void	-	0

1.5. Expressões em C

As expressões em C envolvem normalmente: Constantes, Variáveis e Operadores

1.6 Constantes em C

Representam valores fixos inteiros ou caracteres

1.6.1 Constantes Inteiras

- **Constantes Decimais** → algarismos decimais (0 – 9)
Atenção: não iniciam por 0 Ex. 10 -98 1000005
- **Constantes Octais** → algarismos octais (0 – 7)
Iniciam por 0 Ex. 073 064 01234
- **Constantes Hexadecimais** → algarismos (0 – 9) e (a – f) ou (A – F)
Iniciam por 0x ou 0X Ex. 0x32ad 0X01FF 0X89BD3

As constantes inteiras podem ser modificadas com os tipos unsigned e long

Ex 10U → constante inteira decimal (sem sinal)

010U → constante inteira octal (sem sinal – unsigned)

10000000000L → constante inteira decimal (com sinal e do tipo long)

1.6.2 Constantes de Ponto Flutuante

- constante em precisão simples (float) : 0.023F 1.2e-4F 3.4f
- constante em precisão dupla (double): -0.5e-15 1.24e-8 1.2

1.6.3 Constantes Alfanuméricas

- caractere representado entre apóstrofes Ex: ‘a’ ‘:’ ‘\n’ ‘2’

1.6.4 Constantes “string”

- seqüência de caracteres entre aspas

Ex. “maria” “Av. Sete de Setembro”

1.7 Variáveis em C

- As variáveis armazenam informações que podem ser alteradas ao longo do programa.
- Todas as variáveis devem ser declaradas antes de serem usadas.

1.7.1 Declaração de Variáveis:

- A declaração de uma variável envolve a definição do *tipo* e do *identificador*
- A declaração da variável associa uma área reservada na memória (total de bytes - depende do tipo declarado) ao nome da variável (identificador).

Ex: `int QLAT;` `char Sexo;` `float area;`

1.7.2 Regras para Identificação de Variáveis

- O identificador deve começar por
 Letra: 'a' a 'z' ou 'A' a 'Z'
 ou
 '_' (sublinhado)
- A seguir podem vir letras, '_' e algarismos de 0 a 9

Ex `Total_de_latas,` `_x1,` `Nota2,` `nota2`

- Maiúsculas ≠ Minúsculas
- Não são permitidos nomes ou palavras reservadas

Palavras reservadas em ANSI C					
auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
main	register	return	short	signed	sizeof
static	struct	switch	typedef	union	unsigned
void	volatile	while			

1.8 Operadores em C

1.8.1 Operador de Atribuição: =

Ex: A =10;

Ex:

```
...
float B;
int a,b,c,d;

B = 2.4F;
a=b=c=d=1;
```

1.8.2 Operadores Aritméticos: +, -, *, /, %

Soma: +

Subtração: -

Produto algébrico: *

Ex:

Divisão: /

Resto da divisão: %

```
...
float X;
int A,B,C,D;
A=10;
B=15;
C = A+B-2;
D = B%A;
X = B/A;
X = B/2;
X= B/2.0f;
...
```

1.8.3 Operadores Relacionais: >, <, >=, <=, ==, !=

Maior: >

Menor: <

Maior ou igual: >=

Menor ou igual: <=

Igual: ==

Diferente: !=

Ex:

```
5 > 10
```

```
4 < 2
```

```
'a' == 97
```

```
char a,b;
```

```
a = 'c';
```

```
b = 'c';
```

```
a == b
```

```
'a' == 'b'
```

1.8.3 Operadores Lógicos: &&, ||, !

Interseção (e) : **&&**

União (ou): **||**

Negação (não): **!**

- São usados para combinar expressões relacionais compostas
- Resultam em 0 (falso) e >0 (verdadeiro)

Ex:

```
int contador =1;
char letra = 's';
!( (contador <= 10) && (letra == 'p') )
(letra == 's') || (letra == 'S')
```

1.8.5 Operadores de Incremento e Decremento: ++, --

- Incremento: ++
- Decremento: --

Incremento/decremento pré-fixado : ++x ou --x
incrementa/decrementa e depois usa

Incremento/decremento pós-fixado: x++ ou x--
usa e depois incrementa/decrementa

Ex: **B = 0;**
 A = ++B; /* pré-fixado*/
 C = --A;

B= 0;
 A = B++; /* pós-fixado*/
 C = A--;

Quais os valores de A, B e C após a execução dos comandos sequenciais ?

1.8.6 Operadores Compostos de Atribuição: += -= *= /= %=

I += 2; → I = I+2; X += Y+1; → X = X+Y+1;
X *= 3; → X = X*3; X *= Y+1; → X = X*(Y+1);
Y %= 2; → Y = Y % 2;+

1.9 Operadores e Precedência

Operador	Descrição	Associatividade
⇒ ()	Chamada de função	
⇒ []	referência a elemento de vetor	
⇒ →	ref. a membro de estrutura apontada por um ponteiro	esquerda p/ direita
•	referência a membro de estrutura	
⇒ -	menos (unário)	
⇒ +	mais (unário)	
⇒ ++	incremento	
⇒ --	decremento	
⇒ !	negação lógica	
~	complemento de um	direita p/ esquerda
sizeof (type)	tamanho ocupado por um objeto conversão - cast	
⇒ *	multiplicação	
⇒ /	divisão	esquerda p/ direita
⇒ %	resto da divisão	
⇒ +	adição	
⇒ -	subtração	esquerda p/ direita
<<	deslocamento à esquerda	
>>	deslocamento à direita	esquerda p/ direita
⇒ <	menor que	
⇒ <=	menor ou igual	
⇒ >	maior que	esquerda p/ direita
⇒ >=	maior ou igual	
⇒ ==	igual a	
⇒ !=	diferente de	esquerda p/ direita
&	AND bit a bit	esquerda p/ direita
^	XOR bit a bit	esquerda p/ direita
	OR bit a bit	esquerda p/ direita
⇒ &&	AND Lógico	esquerda p/ direita
⇒	OR Lógico	esquerda p/ direita
? :	condicional	direita p/ esquerda
⇒ =		
⇒ *= /= %=		
⇒ += -= &=		
^= =		
<<= >>=	operadores de atribuição	direita p/ esquerda
,	operador vírgula	esquerda p/ direita

2. Funções de Entrada-Saída em C

entrada padrão → teclado: `leia(..) ⇔ scanf(..)`

saída padrão → tela : `imprima(..) ⇔ printf(..)`

2.1 A função printf

- **Sintaxe da chamada**
`printf("expressão de controle", lista de argumentos);`
- **Expressão de controle**
caracteres impressos na tela + cód. de formatação dos argumentos
- **Lista de argumentos**: expressões, variáveis ou constantes

Algoritmo: `imprima("O valor da variável é",Y);`

Em C: `printf("O valor da variável é %d", Y); /*se Y é int*/`
 ou
`printf("O valor da variável é %f", Y); /*se Y é float*/`

```

/*programa exemplo do uso da função printf*/
#include<stdio.h>
void main( void)
{
    int N=2;
    char Letra ='a';
    float X = 2.5f;
    double Z=3.5e-10;
    printf("O primeiro valor impresso é uma constante decimal %d",15);
    printf("O valor da primeira variável declarada e inicializada é %d",N);
    printf("O valor da segunda variável declarada e inicializada é %c",Letra);
    printf("O valor da primeira variável de ponto flutuante (prec. Simples) é %f",X);
    printf("O valor da segunda variável de ponto flutuante (prec. Dupla) é %f",Z);
    printf("O valor da segunda variável de ponto flutuante (prec. Dupla) é %.11f",Z);
    printf("O valor da expressão que soma 4 ao valor de N é %d", N+4);
    printf("As variáveis utilizadas (declaradas e inicializadas) foram N=%d, \
Letra=%c, X=%f, Z=%.11f", N,Letra,X,Z);
}
  
```

→ Imprime com 6 casas dec.

Código de Formatação do printf()

s c a n f	%c	⇒	Caractere simples (char)
	%i	⇒	Inteiro com sinal (int)
	%d	⇒	Inteiro Decimal (int)
	%o	⇒	Inteiro Octal (int)
	%x	⇒	Inteiro Hexadecimal (int)
	%u	⇒	Inteiro sem sinal (unsigned)
	%ld	⇒	Decimal longo
	%f	⇒	Ponto flutuante (float e double)
	%lf	⇒	Ponto flutuante longo (double)
	%e	⇒	Notação científica (mantissa e expoente)
	%g	⇒	%e ou %f (o mais curto)
	%s	⇒	Cadeia de caracteres
	%%	⇒	imprime o sinal de porcentagem

Códigos Especiais

Caractere	Significado	Valor ASCII (dec)
\a	⇒ Campainha	7
\b	⇒ Retrocesso (Backspace)	8
\t	⇒ TAB - horizontal	9
\v	⇒ TAB - vertical	11
\n	⇒ Nova Linha	10
\f	⇒ Salta Página de Formulário	12
\r	⇒ Retorno do Carro	13
\”	⇒ Aspas	34
\’	⇒ Apóstrofo (’)	39
\?	⇒ Interrogação (?)	63
\\	⇒ Barra Invertida (\)	92
\0	⇒ Nulo	0

```

/*****
Programa exemplo de constantes e variaveis em C
ilustra a formatacao da funcao printf para
alguns tipos da dados basicos
*****/

#include <stdio.h> /* inclui definicoes basicas p/ I/O */

#define CONST_EX 50 /* nomeacao da constante 50 */

void main(void )
{
    int vi; /* declaracao das variaveis */
    char vc;
    float vx;
    double vy;

    vi = CONST_EX; /* atribuicao de valores */
    vc = 'm';
    vx = -10.3f;
    vy = 0.023;

    printf("Exemplo de constante  CONST_EX = %d \n",CONST_EX);
    printf("Exemplo de variavel int          vi = %i \n", vi);
    printf("Exemplo de variavel char         vc = %c \n", vc);
    printf("Exemplo de variavel float        vx = %f \n", vx);
    printf("Exemplo de variavel double        vy = %f \n", vy); /* ou %lf */

    vi = 10;
    printf("variavel int (decimal) vi = %d \n",vi);
    printf("variavel int (hexadecimal) vi = %x \n",vi);
    printf("variavel int (octal) vi = %o \n",vi);
}

```

Printf() com Inteiros

```
#include <stdio.h>
void main(void)
{
    int vari = 2;
    float varx = 2.4f;

    printf("Exemplos : vari = %d, varx = %f \n", vari, varx);
    printf("A porcentagem da população é %d %% \n", 85);
}
```

- saída do programa
Exemplos : vari = 2, varx = 2.4
A porcentagem da população é 85 %

Tamanho de Campos

Ex: printf("total = %2d \n", 100);
printf("total = %4d \n", 100);
printf("total = %5d \n", 100);

- saída :
total = 100
total = 100
total = 100

Complementos com zeros à esquerda

Ex: printf("total = %04d \n", 21);
printf("total = %06d \n", 21);

saída : total = 0021

- total = 000021

Printf() com Ponto Flutuante

```
#include <stdio.h>
void main( void)
{
    printf(“%4.2f\n”,3456.78f);
    printf(“%3.2f\n”,3456.78f);
    printf(“%3.1f\n”,3456.78f);
    printf(“%10.3f\n”,3456.78f);
}
```

- saída do programa

```

| 3456.78
| 3456.78
| 3456.8
| 3456.780
```

Alinhamento à Direita

Ex:

```
printf(“%10.2f %10.2f %10.2f\n”,8.0,15.3,584.13);
printf(“%10.2f %10.2f %10.2f\n”,834.0,1500.55,4890.21);
```

- saída :

```

      .00      15.30      584.13
834.00    1500.55    4890.21
```

Alinhamento à Esquerda

Ex:

```
printf(“%-10.2f %-10.2f %-10.2f\n”,8.0,15.3,584.13);
printf(“%-10.2f %-10.2f %-10.2f\n”,834.0,1500.55,4890.21);
```

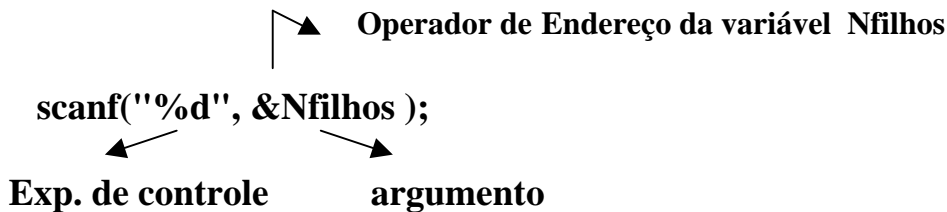
- saída :

```

8.00      15.30      584.13
834.00    1500.55    4890.21
```

2.2 A função scanf

- **Sintaxe da chamada**
`scanf("expressão de controle", lista de argumentos);`
- **Expressão de controle**
 caracteres lidos do teclado + cód. de formatação dos argumentos
- **Lista de argumentos:** endereços das variáveis a serem lidas



Algoritmo: leia(A,B);

Em C: `scanf("%d %d",&A, &B); /*se A e B são int */`
 ou
`scanf("%f %f",&A, &B); /*se A e B são float */`

```

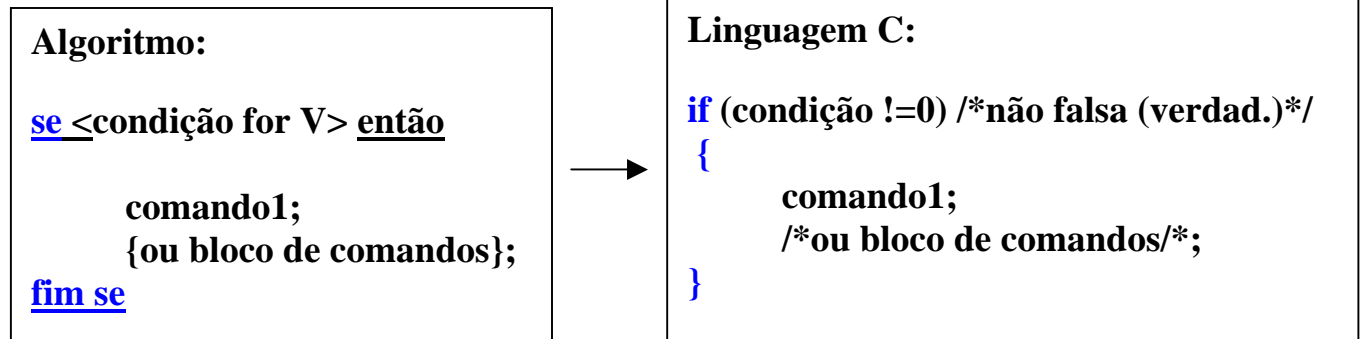
/*programa exemplo do uso da função scanf*/
#include<stdio.h>
void main()
{
    int N;
    char Letra;
    float X;
    double Z;

    printf("Entre com o valor da var. N (valor inteiro decimal): ");
    scanf("%d",&N);
    printf("Entre com o valor da var. Letra (valor alfanumérico): ");
    scanf("\n%c",&Letra);
    printf("Entre com o valor da var. X (valor de pto. flutuante): ");
    scanf("%f",&X);
    printf("Entre com o valor da var. Z (valor de pto. flutuante): ");
    scanf("%lf",&Z);
    printf("As variáveis utilizadas (declaradas e lidas) foram N=%d, Letra=%c,
X=%f, Z=%f", N,Letra,X,Z);
}

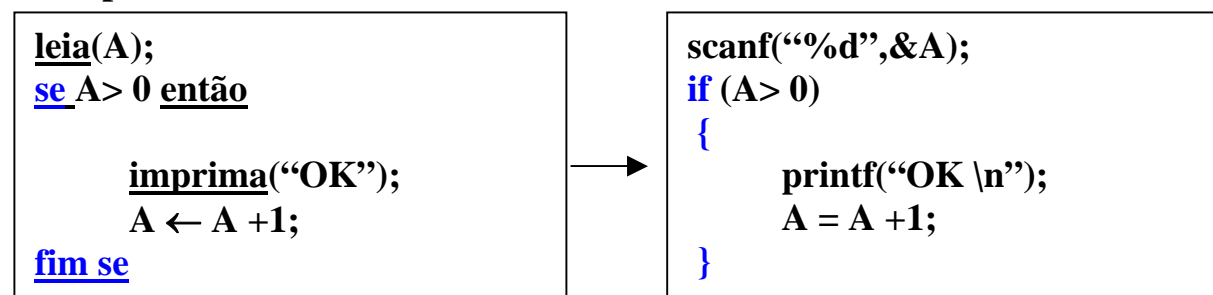
```

3. Estruturas de Seleção em C

3.1 Seleção Simples



Exemplos:



```
/* Exemplo de um programa com seleção simples */
#include <stdio.h>
void main( void )
{
    char entrada;
    scanf("%c",&entrada);
    if (entrada == 'f' || entrada == 'F')    //aqui NÃO tem ponto-e-vírgula
    {
        printf("Voce pressionou a tecla para finalizar \n");
        printf("Fim do programa");
    }
    printf("O valor da variável fornecida é %c", entrada);
}
```

3.2 Seleção Composta

Algoritmo:

```
se <condição for V> então
    comando1; {ou bloco}
senão
    comando2; {ou bloco}
fim se
```

Linguagem C:

```
if (condição !=0) /* não falsa*/
{
    comando1; /*ou bloco*/
}
else
{
    comando2; /* ou bloco*/
}
```

Exemplos:

```
leia(A);
se A > 0 então
    imprima("OK");
    A ← A +1;
senão imprima("Erro");
fim se
```

```
scanf("%d",&A);
if (A > 0)
{
    printf("OK \n");
    A = A +1;
}
else printf("erro");
/* como neste caso o else só tem 1
comando não precisa de chaves*/
```

Exemplo de um programa com seleção composta:

```
#include <stdio.h>
void main( void )
{
    char entrada;
    scanf("%c",&entrada);
    if (entrada == 'f' || entrada == 'F') // aqui NÃO tem ponto-e-vírgula
    {
        printf("Voce pressionou a tecla para finalizar \n");
        printf("Fim do programa");
    }
    else // aqui NÃO tem ponto-e-vírgula
    {
        printf("Voce NAO digitou a tecla para finalizar \n");
        printf("Continuando a execução do programa \n");
    }
    printf("O valor da variável fornecida é %c", entrada);
}
```

3.3 Seleção Encadeada

início {Algoritmo para o Cálculo do maior e menor número de uma série de números inteiros positivos}

inteiro: MAIOR, MENOR, VALOR;

leia(VALOR);

MAIOR ← VALOR;

MENOR ← VALOR;

enquanto VALOR ≠ 0 **faça**

se VALOR > MAIOR **então**

 MAIOR ← VALOR;

senão

se VALOR < MENOR **então**

 MENOR ← VALOR;

fim se

fim se

leia(VALOR);

fim enquanto;

escreva (MAIOR, MENOR);

fim

#include <stdio.h> /*Algoritmo para o Cálculo do maior e menor número de uma série de números inteiros positivos*/

void main(void)

{

int MAIOR, MENOR, VALOR;

scanf("%d", &VALOR);

 MAIOR = VALOR;

 MENOR = VALOR;

while (VALOR != 0) ←

{

if(VALOR > MAIOR) ←

 MAIOR = VALOR;

else if (VALOR < MENOR) ←

 MENOR = VALOR;

scanf("%d", &VALOR);

}

printf(" %d %d\n", MAIOR, MENOR);

}

/* Neste programa todos os Ifs (e seus elses) têm apenas um comando, então as { ...} são desnecessárias */

aqui NÃO tem
ponto-e-vírgula

3.4 Seleção de Múltipla Escolha

escolha X

```

caso E1: Comando1; {ou bloco}
caso E2: Comando2; {ou bloco}
    :
caso EN; ComandoN; {ou bloco}
caso contrário: ComandoM; {ou bloco}
fim escolha

```

switch(X)

```

{
  case E1: Comando1; {ou bloco}; break;
  case E2: Comando2; {ou bloco}; break;
    :
  case EN: ComandoN; {ou bloco}; break;
  default: ComandoM; {ou bloco}; break;
}

```

Exemplos:

Início {Algoritmo para criação e teste de um MENU}

caractere: op;

leia(op);

escolha(op)

caso 'c' : imprima("copiando arquivo"); {comandos p/ copiar}

caso 'a' : imprima("apagando arquivo"); {comandos p/ apagar}

caso 'd' : imprima("criando diretório"); {comandos p/ criar dir}

caso 'f' : imprima("formatando disquete"); {comandos p/ formatar}

caso contrário: imprima("saindo do programa"); {comandos para

fim escolha



fim

```

#include <stdio.h> /* programa para criação e teste de um MENU*/

```

```

void main (void )

```

```

{   char op;

```

```

    printf("Entre com a opção:");

```

```

    scanf("%c", &op);

```

```

    switch (op)

```

```

    {

```

```

        case 'c' : printf("copiando arquivo\n"); /*comandos p/ copiar*/;
                break;

```

```

        case 'a' : printf ("apagando arquivo\n"); /*comandos p/ apagar*/;
                break;

```

```

        case 'd' : printf ("criando diretório\n"); /*comandos p/ criar dir*/;
                break;

```

```

        case 'f' : printf ("formatando disquete\n"); /*comandos p/ formatar*/;
                break;

```

```

        default: printf("saindo do programa\n");
                break;

```

```

    }

```

```

}

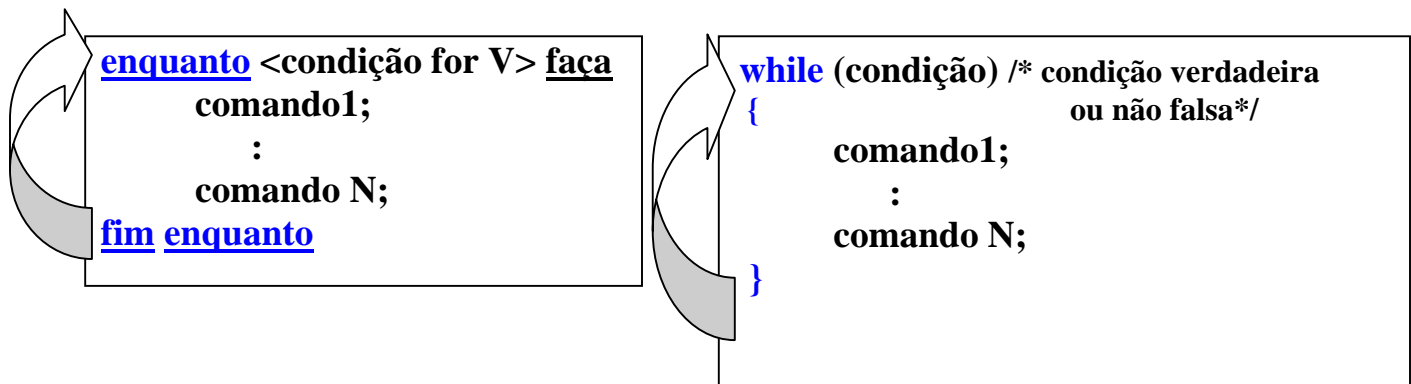
```

aqui NÃO tem
ponto-e-vírgula

4. Estruturas de Repetição em C

4.1 Repetição com teste no início do laço

Estrutura: enquanto \Leftrightarrow while



Exemplo: A variável de controle deve ser **inicializada antes do início** e **modificada dentro do laço**

```

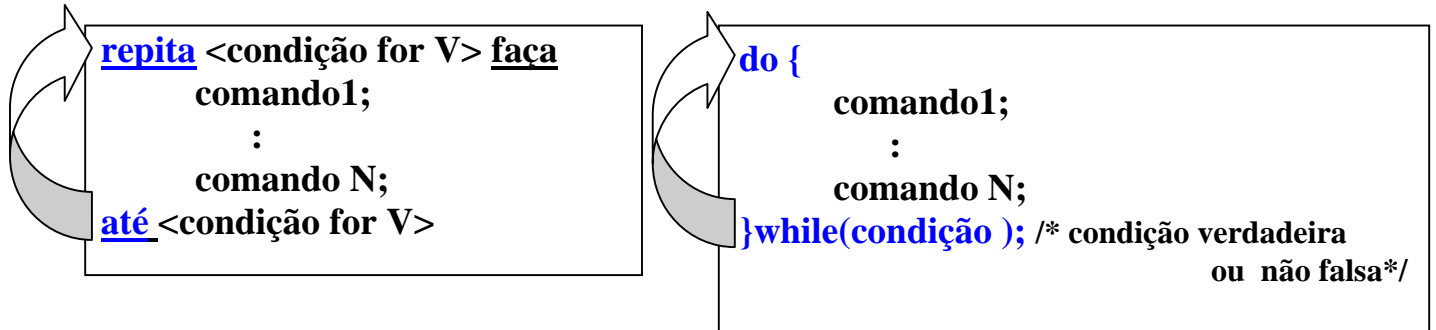
/*programa que imprime o quadrado e o cubo de uma série de valores
inteiros até que se digite 0*/
#include<stdio.h>
#include<math.h>
void main ( void )
{
    int I;

    printf("Entre com um valor inteiro: ");
    scanf("%d",&I); /* var. de controle inicializada*/
    while(I!=0) /* condição de execução: var de controle I≠0 */
    {
        printf("%d | %f | %f\n", I, pow(I,2), pow(I,3) );
        printf("Entre com um valor inteiro: ");
        scanf("%d",&I); /* var de controle modificada*/
    }
}

```

4.2 Repetição com teste no fim do laço

Estrutura: repita \Leftrightarrow do-while /* faça-enquanto*/



Exemplo: A variável de controle pode ser **modificada dentro do laço**, mas como o teste é feito no fim, a **condição de parada não deve ser executada**


```

/*programa que imprime o quadrado e o cubo de uma série de valores
inteiros até que se digite 0*/
#include <math.h>
#include<stdio.h>
void main ( void)
{
    int I;

    do
    {
        printf("Entre com um valor inteiro: ");
        scanf("%d",&I); /* var de controle modificada*/
        if (I != 0)
            printf("%d | %f | %f\n", I, pow(I,2), pow(I,3) );
    } while(I!=0); /* condição de execução: I≠0 */
}


```

4.3 Repetição com variável de controle incremental



```


para Variável de ValInic até ValFin passo P faça
    comando1;
    :
    comando N;
fim para
  
```



```

for (Variável = ValInic; Variável <= ValFin; Variável = Variável +P)
{
    comando1;
    :
    comando N;
}
  
```

Exemplo:



```

/*programa que imprime o quadrado e o cubo de
conjunto de 100 valores inteiros */
#include<math.h>
#include<stdio.h>
void main ( void )
{
    int C, I;
    for (C=1;C<=100;C=C+1) ← aqui NÃO tem ponto-e-vírgula
    {
        printf("Entre com um valor inteiro: ");
        scanf("%d",&I);
        printf("%d | %f | %f \n", I, pow(I,2), pow(I,3) );
    }
}
  
```

Algoritmo -> C : Estruturas de Repetição

Estrutura Enquanto - Faça

inicio

real : MA, {média anual de um dado aluno}
 ACM, {acumulador}
 MAT; {media anual da turma}

inteiro : CON; {contador}

ACM ← 0;

CON ← 0;

leia (MA);

enquanto MA ≠ -1 faça {teste da condição, se e' falsa -> termina laço}

ACM ← ACM + MA;

CON ← CON + 1;

leia (MA);

fim enquanto

MAT ← ACM / CON;

imprima(" A media anual da turma de",CON,"alunos é ",MAT);

fim

Laços de Repetição : While (Linguagem C)

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

```
void main(void)
```

```
{
```

```
float MA, /*média anual de um dado aluno */
```

```
ACM, /* acumulador */
```

```
MAT; /* media anual da turma */
```

```
int CON; /* contador */
```

```
ACM = 0;
```

```
CON = 0;
```

```
scanf ( "%f",&MA);
```

```
while (MA != -1) /*teste da condição */
```

```
{
```

```
ACM = ACM + MA;
```

```
CON = CON + 1;
```

```
scanf ( "%f",&MA);
```

```
}
```

```
MAT = ACM / CON;
```

```
printf ("media anual da turma de %d alunos = %2.1f ",CON,MAT);
```

```
}
```

Estrutura Repita - Até

inicio

real : MA, {média anual de um dado aluno}
 ACM, {acumulador}
 MAT; {media anual da turma}
inteiro : CON; {contador}

ACM ← 0;
 CON ← 0;

repita

leia (MA);
se MA ≠ -1
então
 ACM ← ACM + MA;
 CON ← CON + 1;

fim se

até MA = -1 {teste da condição de parada- se condição é verdadeira -> finaliza o laço}
 MAT ← ACM / CON;
imprima(" A media anual da turma de",CON,"alunos é ",MAT);

fim

Laços de Repetição : Do-While (Linguagem C)

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

```
void main(void)
```

```
{ float MA, /*média anual de um dado aluno */  
  ACM, /* acumulador */  
  MAT; /* media anual da turma */  
  int CON; /* contador */
```

```
  ACM = 0;  
  CON = 0;
```

```
  do
```

```
  {
```

```
    scanf ( "%f",&MA);  
    if (MA != -1)  
    {  
      ACM = ACM + MA;  
      CON = CON + 1;
```

```
    }  
  } while (MA != -1); /*teste da condição - se condição falsa -> abandona*/  
  MAT = ACM / CON;  
  printf ("media anual da turma de %d alunos = %2.1f ",CON,MAT);}
```

Estrutura Para - Faça

inicio

real : MA, {média anual de um dado aluno}
 ACM, {acumulador}
 MAT; {media anual da turma}

inteiro : CON; {contador}

ACM ← 0;

para CON de 1 até CON<=50 passo 1 faça

leia (MA);

ACM ← ACM + MA;

fim para

MAT ← ACM / CON;

imprima(" A media anual da turma de",CON,"alunos é ",MAT);

fim

Laços de Repetição : for (Linguagem C)

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

```
void main(void)
```

```
{
```

```
    float MA, /*média anual de um dado aluno */
```

```
    ACM, /* acumulador */
```

```
    MAT; /* media anual da turma */
```

```
    int CON; /* contador */
```

```
    ACM = 0;
```

```
    for(CON = 1; CON <= 50; CON = CON + 1)
```

```
    {
```

```
        scanf ( " %f ",&MA);
```

```
        ACM = ACM + MA;
```

```
    }
```

```
    MAT = ACM / 50;
```

```
    printf ("media anual da turma de %d alunos = %2.1f ",50,MAT);
```

```
}
```

5. Estruturas de Dados em C

- **Tipos Primitivos de Dados em C:**

char, int, float, double

- **As estruturas de dados definem como os tipos primitivos serão organizados.**

Exemplos:

Vetores, Matrizes, Registros, Listas, Filas, etc...

O programa a seguir calcula a média das notas para uma turma de 40 alunos:

```
#include<stdio.h>
void main ( void )
{
    float MedAlun, AcumMed, MedTur;
    int ContAlun;

    AcumMed = 0;
    for(ContAlun=1;ContAlun<=40;ContAlun++)
    {
        scanf(“%f”, &MedAlun); /*le a media de cada aluno*/
        AcumMed = AcumMed + MedAlun;
    }
    MedTur = AcumMed/40.0f;
    printf(“A media da turma de 40 alunos e = %.2f\n”,MedTur);
}
```

- **Como alterar o algoritmo para que o total de alunos N(conhecido previamente pelo usuário) possa ser fornecido ?**
- **Como calcular o total de alunos acima da média ?**
- **Como entrar com o nome de cada aluno ?**
- **Como guardar as parciais de cada aluno para o calculo da media de cada aluno e depois o calculo da media da turma ?**

- **Como calcular o total de alunos acima da média ?**

Para calcular a quantidade de notas acima da média é necessária a comparação da média de cada aluno MedAlun com a variável MedTur.

Observações:

a) no desenvolvimento do programa sem o uso de vetores, as médias de cada aluno foram guardadas em uma única variável MedAlun.

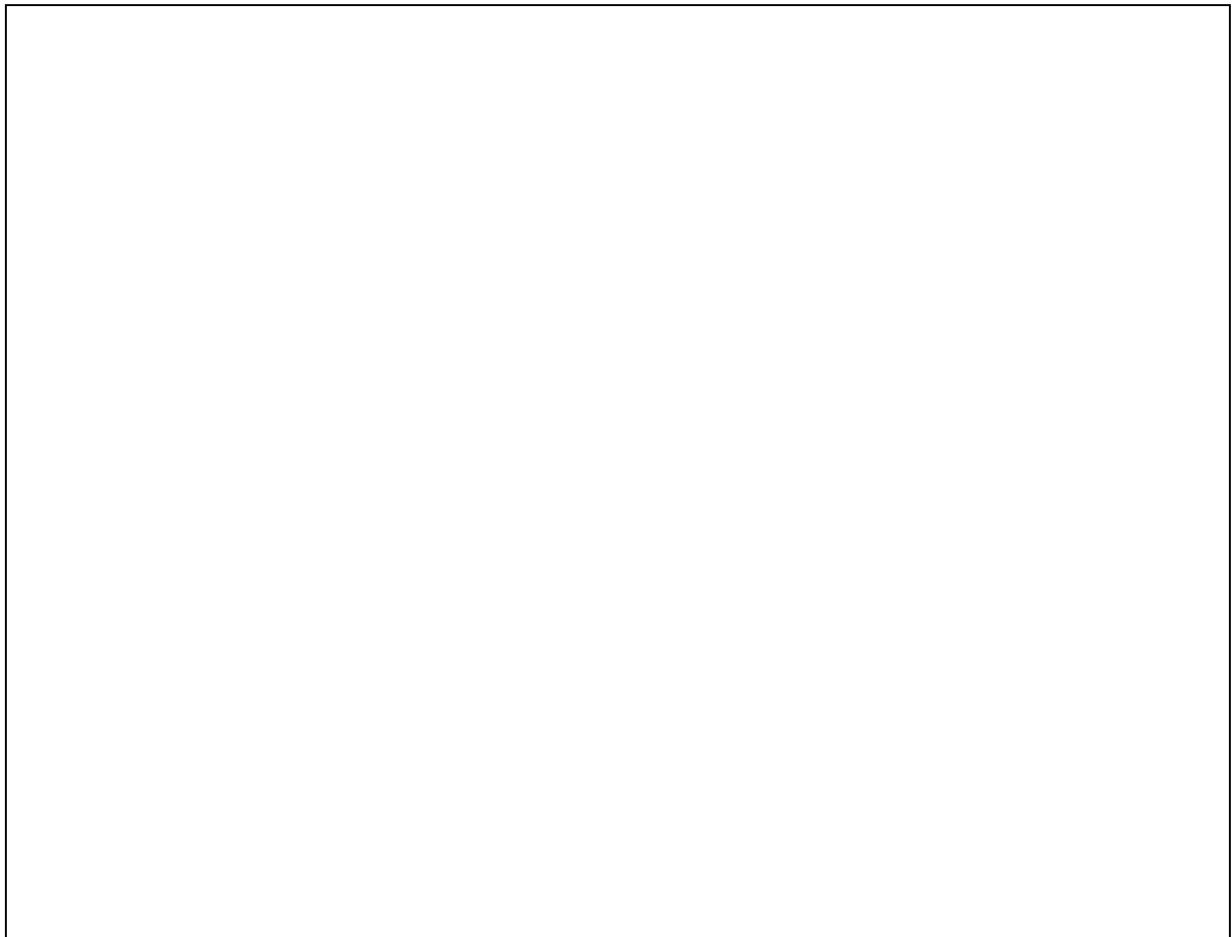
b) Ao final do processo de repetição só teremos o valor da última média lida (MedAlun para o 40º aluno ou N-ésimo aluno)

Como fazer para solucionar este problema ?

Solução 1) utilizar uma variável para cada aluno

Obs. inviável para um número fixo grande de alunos ou quando se quer alterar o total de alunos a cada execução

Solução 2) utilizar um vetor para guardar as médias de cada aluno e depois de calculada a MedTur comparar esta com cada média que foi guardada no vetor

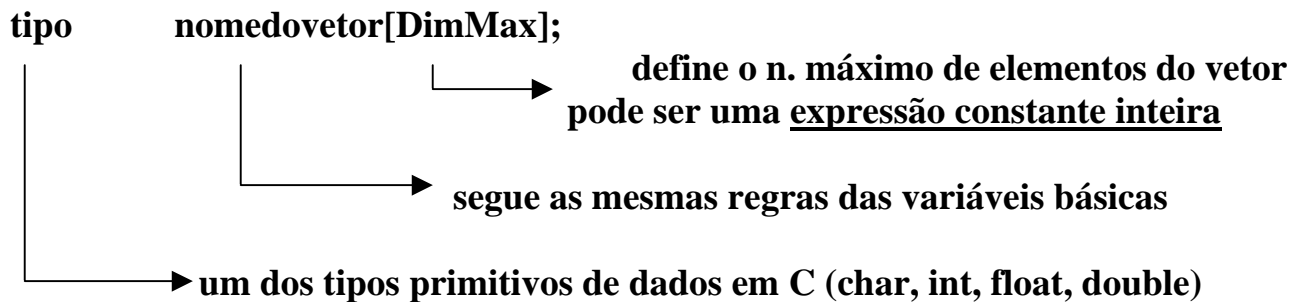


5.1. Vetores - Estruturas de Dados Homogêneas Unidimensionais

Os vetores são variáveis compostas *unidimensionais* que armazenam um conjunto de *dados do mesmo tipo*

Necessitam de apenas um índice de acesso.

Declaração:



Em C, os índices dos vetores variam de 0 a DimMax-1

Exemplo

dclaração: float NOTAS[100];
int Conjunto[50];
char Nome[15];

Acesso:

NOTAS[0]	Conjunto[0]	Nome[0]
NOTAS[1]	Conjunto[1]	Nome[1]
.	.	.
.	.	Nome[14]
.	Conjunto[49]	
.		
NOTAS[99]		

5.1.1 Vetores Numéricos

Recebem valores inteiros, de ponto flutuante (prec. Simples) e de ponto flutuante (prec. Dupla)

- Declaração e Inicialização (feitas conjuntamente)

```
int Vet[4] = {0,0,0,0}; /* inicializa todos com 0*/
```

```
int Vet[4] = {-1,-1}; /* inicializa os dois primeiros elementos com -1*/
```

```
float V[3] = {1.0f, 1.1f, 1.5f}; /* inicializa todos com const. tipo float*/
```

```
int A[ ] = {0,0,0,0,0,0,0,0}; /* a dimensão assume o tamanho da inic.*/
```

A declaração e inicialização conjuntas é útil para vetores de dimensão reduzida

- Atribuição

As atribuições devem ser feitas elemento a elemento

1. Atribuição feita pelo programa

<pre>int vetorA[10]; VetorA[0] = 0; VetorA[1] = 0; ... VetorA[9] = 0;</pre>	ou	<pre>int C, vetorA[10]; for (C=0;C<=9;C++) vetorA[C] = 0;</pre>
--	----	---

2. Atribuição feita pelo usuário

<pre>int vet[10]; scanf("%d",&vet[0]); ... scanf("%d",&vet[9]);</pre>	ou	<pre>int C, vet[10]; for (C=0;C<=9;C++) scanf("%d",&vet[C]);</pre>
--	----	--

5.1.2 Vetores de Caracteres: Strings

As variáveis do tipo vetor de caracteres (strings) são declaradas como

```
char vet[DimMax];
```

Recebem um conjunto de valores do tipo char denominados constantes strings
Ex. “Maria”, “Av. 7 de Setembro”.

Todo vetor de caracteres (string) em C deve terminar com o caractere nulo ‘\0’ que indica o fim da string (nenhum caractere após este é considerado)

- Declaração e inicialização (feitas conjuntamente)

```
char Nome[6] = {'M','a','r','i','a','\0'};
```

```
char Ender[20] = {'A','v','.',',','7',' ','d','e',' ','S','e','t','e','m','b','r','o','\0'};
```

```
char cidade[ ] = “Curitiba”;
```

- Atribuição

As atribuições podem ser feitas elemento a elemento

1. Atribuição elemento a elemento feita pelo programa

```
char fruta[5];
fruta[0] = 'p';
fruta[1] = 'e';
fruta[2] = 'r';
fruta[3] = 'a';
fruta[4] = '\0';
```

ou

```
int C;
char vetorABCD[27];

for (C=0;C<=25;C++)
    vetorABCD[C] = C+97;
vetorABCD[26] = '\0';
```

2. Atribuição elemento a elemento feita pelo usuário

```
char Nome[6];

scanf(“\n%c”,&Nome[0]);
...
scanf(“\n%c”,&Nome[4]);
Nome[5] = '\0';
```

```
int C;
char vetorLetras[10];
for (C=0;C<=8;C++)
{
    scanf(“\n%c”,&vetorLetras[C]);
}
vetorLetras[C] = '\0';
```

```

/* Programa Exemplo de vetores de inteiros e vetores de caracteres (strings)
Programa lê 3 notas e o nome do aluno (caractere a caractere)
e Calcula e Imprime(caractere a caractere) o nome e a média das 3 notas */

#include <stdio.h>
void main (void)
{
    char letra_nome[15]; /* 1 4 letras + '\0'*/
    float nota[3], soma, media;
    int indice;

    soma = 0.0f;
    printf("Entre com as notas do aluno \n");
    for(indice=0; indice<3;indice++)
    {
        scanf("%f", &nota[indice]);
        soma = soma+nota[indice];
    }
    media = soma/3.0f;
    printf("Entre com as letras do nome do aluno \n");
    for(indice=0; indice<15;indice++)
    { /* lê uma letra de cada vez (o usuário TEM que entrar com 15 letras)*/

        scanf("\n%c", &letra_nome[indice]);

    }
    letra_nome[15] = '\0';
    printf("A media do(a) aluno(a) ");
    for(indice=0; indice<15;indice++)
        printf("%c", letra_nome[indice]);
    printf("foi %.2f\n",media);
}

```

Desvantagem: como fazer quando os nomes tiverem menos ou mais de 15 letras ?

As strings permitem que, ao invés de se atribuir um valor a um elemento de cada vez, a atribuição seja feita num único passo.

3. Atribuição de todo o vetor feita pelo programa

Este tipo de atribuição pode ser feito pela função `strcpy(...)`; cujo protótipo está em `<string.h>`

Exemplo:

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

void main( void )
{
    char fruta[5], vetorABCD[27], Nome[15], Ender[50];

    strcpy(fruta,"pera"); /* equiv a fruta = "pera" */
    strcpy(Nome,"Maria"); /* equiv a Nome = "Maria" */
    strcpy(Ender,"Av. 7 de Setembro"); /* equiv a Ender = "Av. 7 de Setembro" */
    strcpy(vetorABCD,"abcdefghijklmnopqrstuvwxyzyz");

    /* a função strcpy já inclui o '\0' no final*/
    printf("A fruta e %s \n O nome e %s \n",fruta,Nome);
    printf("O endereço e %s \n e o abecedário e %s \n",Ender,vetorABCD);

}
```

4. Atribuição de todo o vetor feita pelo usuário

→ `gets(Nome_vetor) /* mais utilizada */`

ou

```
scanf("%s",&Nome_vetpr[0]);
ou scanf("%s",Nome); /* menos utilizadas*/
```

/ as funções scanf com %s e gets já incluem o '\0' no final mas scanf não aceita espaço porque considera-o como finalizador (elimina o que vier depois)*/*

Exemplos: `gets(fruta);` `gets(Nome);` `gets(Ender);`

```
strcpy(fruta,"pera");   ≠   gets(fruta);
```

```

/* Alteracao do programa anterior para ler e imprimir como string */
#include <stdio.h>
#include<string.h>
void main (void)
{
    char nome[51]; /* até 50 letras + '\0'*/
    float nota[3], soma, media;
    int indice;

    soma = 0.0f;
    printf("Entre com as notas do aluno \n");
    for(indice=0; indice<3;indice++)
    {
        scanf("%f", &nota[indice]);
        soma = soma+nota[indice];
    }
    media = soma/3.0f;
    printf("Entre com o nome do aluno (max de 4 letras) \n");
    scanf("%s"); /* limpar memória do teclado para ler character*/
    gets(nome);
    printf("A media do(a) aluno(a) %s foi %.2f\n",nome,media);
}

```

Funções de manipulação de strings:

- **leitura:**
scanf("%s",varstr); /* protótipo em <stdio.h> **NÃO** aceita espaço*/
gets(varstr); /* protótipo em <string.h> */
- **impressão:**
printf("%s \n",varstr); /* protótipo em <stdio.h> */
puts(varstr); /* protótipo em <string.h> **imprime e já salta linha***

manipulação: /* todas com protótipo em <string.h> */

strcpy(varstr,str2); **copia** a constante ou variável string str2 na variável varstr

strlen(str); **calcula o tamanho** (sem contar o '\0') da var. ou constante string

strcat(varstr1,str2); **une** as duas strings varstr1+str2

strcmp(str1,str2) **compara** as duas strings e retorna

<p>< 0 se str1 < str2 (ordem alfab)</p> <p>= 0 se str1 = str2</p> <p>> 0 se str1 > str2 (ordem alfab)</p>
--

Como guardar as parciais (total de 2) de cada aluno para o cálculo da média do aluno e depois o cálculo da média da turma ?

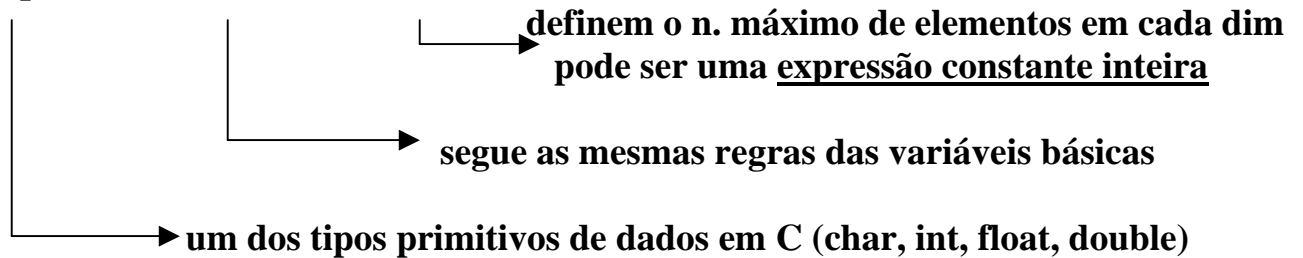
5.2. Matrizes - Estruturas de Dados Homogêneas Multidimensionais

As matrizes são variáveis compostas *multidimensionais* que armazenam um conjunto de *dados do mesmo tipo*

Necessitam de mais de um índice de acesso.

Declaração:

tipo NomeDaMatriz[DimMax1] [DimMax2]... [DimMaxN];



Em C, os índices das matrizes variam de [0 a DimMax1-1], [0 a DimMax2-1] ...

Exemplo

declaração: float NOTAS[100][100];

Acesso:

NOTAS[0][0]

NOTAS[0][1]

.

NOTAS[0][99]

NOTAS[1][0]

NOTAS[1][1]

.

NOTAS[1][99]

.

.

NOTAS[99][0]

.

NOTAS[99][99]

5.2.1 Matrizes Numéricas

Recebem valores do tipo int, float, double

- Declaração e Inicialização (feitas conjuntamente)

```
int Mat[2][4] = {{0,0,0,0},
                {0,0,0,0}}; /* inicializa todos com 0*/
```

```
int MAT[3][5] = {{-1,-1,-1,-1,-1}}; /* inicializa só a 1ª. linha com -1*/
```

```
float M[2][2] = {{1.0f, 1.1f},{1.5f,-0.9f}}; /* inicializa com const. tipo float*/
```

```
int A[ ][ ] = {{0,0,0},{0,0,0}}; /* a dimensão assume o tamanho da inic.*/
```

útil para matrizes de dimensão reduzida

- Atribuição

As atribuições devem ser feitas elemento a elemento

1. Atribuição feita pelo programa

```
int matA[2][5];
matA[0][0] = 0;
matA[0][1] = 0;
...
matA[0][4] = 0;
matA[1][0] = 0;
...
matA[1][4] = 0;
```

ou

```
int L,C, matA[2][5];
for (L=0;L<=1;L++)
  for (C=0;C<=4;C++)
    matA[L][C] = 0;
```

2. Atribuição feita pelo usuário

```
int matA[2][5];

scanf("%d",&matA[0][0]);
...
scanf("%d",&matA[1][4]);
```

```
int L,C, matA[2][5];

for (L=0;L<=1;L++)
  for (C=0;C<=4;C++)
    scanf("%d",&matA[L][C]);
```

5.2.2 Matrizes de Caracteres (Matriz de Strings)

As variáveis do tipo matriz de caracteres (strings) são declaradas como

```
char vet[DimMax1][DimMax2] ... [DimMaxN];
```

Recebem (em cada linha) uma string de caracteres

Cada linha da matriz de caracteres deve terminar com o caractere nulo '\0'

- Declaração e inicialização (feitas conjuntamente)

```
char Nome[3][6] = {{'M','a','r','i','a', '\0'}, {'J','o','e','l', '\0'}, {'A','n','a', '\0'}};
```

```
char Ender[2][21] = {
    {'A','v','.',',','7',' ','d','e',' ','S','e','t','e','m','b','r','o','\0'},           {'R','u',
    {'P','e','r','n','a','m','b','u','c','o','\0'}                                     } ;
```

```
char cidades[2][51] = {"Curitiba","Campinas"};
```

- Atribuição

As atribuições podem ser feitas elemento a elemento

1. Atribuição elemento a elemento feita pelo programa

```
char frutas[3][5];
frutas[0][0] = 'p';
frutas[0][1] = 'e';
frutas[0][2] = 'r';
frutas[0][3] = 'a';
frutas[0][4] = '\0';
frutas[1][1] = 'u';
```

ou

Matriz frutas

p	e	r	a	\0
u	v	a	\0	

2. Atribuição elemento a elemento feita pelo usuário

```
char Nome[4][6];

scanf("\n%c",&Nome[0][0]);
...
scanf("\n%c",&Nome[0][4]);
```

ou

```
char Letras[5][10];
int L,C;
for (L=0;L<=4;L++)
    { for (C=0;C<=8;C++)
        {
            scanf("\n%c",&Letras[L][C]);
        }
        Letras[L][C] = '\0';
    }
```

Ou as atribuições podem ser feitas em cada linha da matriz

3. Atribuição de **cada linha** feita pelo programa

```
char frutas[3][5];

strcpy(frutas[0], "pera"); /* equiv a frutas[0] = "pera" */
strcpy(frutas[1], "uva"); /* equiv a frutas[1] = "uva" */
strcpy(frutas[2], "kiwi"); /* equiv a frutas[2] = "kiwi" */
/* lembrando que o protótipo de strcpy(..) está em string.h*/
```

4. Atribuição de **cada linha** feita pelo usuário

```
char Nomes[4][6]

scanf("\n");
gets(Nomes[0]);
...
scanf("\n");
gets(Nomes[3]);
```

ou

```
char Nomes[5][31];
int J;
for (J=0; J<=4; J++)
{
    printf("Entre com o nome (máx. de 30 caracteres)\n");
    fflush(stdin);
    gets(Nomes[J]);
}
```






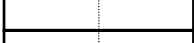

INFORMATIVO

Exemplo Alocação de Vetores e Matrizes

Declaração de Vetores e Esquemático da Memória

DECLARAÇÃO

MEMÓRIA ALOCADA

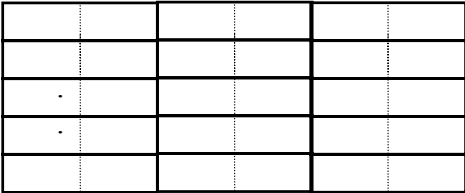

int a;	a		(2 bytes)
float b;	b		(4 bytes)
int vetor[5];	vetor[0]		(2 bytes)
	vetor[1]		(2 bytes)
	.		(2 bytes)
	.		(2 bytes)
	vetor[4]		(2 bytes)

Total Alocado : $2 + 4 + (2 * 5) = 16$ bytes

Declaração de Matrizes e Esquemático da Memória

DECLARAÇÃO

MEMÓRIA ALOCADA

int mat[5][3];		
		(3* 2 bytes)
		(3* 2 bytes)
		(3* 2 bytes)
		(3* 2 bytes)
		(3* 2 bytes)

Total Alocado : $5 * (3 * 2) = 30$ bytes

6. Modularização - Funções em C

As funções facilitam a elaboração e o entendimento de programas em C

Como exemplo das facilidades que as funções podem trazer, considere o programa em C que calcula o fatorial de um número inteiro fornecido pelo usuário:

```
#include <stdio.h>
void main(void)
{   int N,Numero;
    long int Fat;

    printf("Entre com o valor do Numero");
    scanf("%d",&Numero);
    N=Numero;
    if(N<0)
        printf("Não existe fatorial de %d (valor negativo) \n",Numero);
    else{
        if(N==0 )
            Fat = 1; /* fatorial de 0 é por definição igual a 1 */
        else
        {   Fat = N;          /*início do cálculo do fatorial*/
            while(N > 1)
            {
                Fat = Fat*(N-1);
                N=N-1;
            } /*fim do cálculo do fatorial*/
        }
        printf("O fatorial de %d e igual a %ld \n",Numero,Fat);
    }
}
```

Como calcular a combinação de N elementos P a P ? $C_P^N = \frac{N!}{P!(N-P)!}$

Solução 1) Via repetição de código (menos indicada)

Solução 2) Utilizando uma função para o cálculo do fatorial

Comb = FuncFat(N) / (FuncFat(P)*FuncFat(N-P));

/* **Solução 1:** Programa para o cálculo de Combinação de N elementos tomados P a P
Com repetição de código (Sem utilização de Módulos) */

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

```
void main( void )
{
    int N,P,X;
    long int FAT1, FAT2, FAT3;
    int COMB;

    scanf(“%d %d”,&N,&P);
    if (N < P)
        printf(“Combinação indefinida\n “);
    else if(N < 0 || P < 0)
        printf(“Não existe cominação de números negativos\n”);
    else
    {
        X = N; /*Calcula Fatorial de N em FAT1*/
        if (X == 0)
            FAT1 = 1;
        else {FAT1 = X;
            while (X > 1)
            {
                FAT1 =FAT1 * (X-1);
                X--;
            }
        }

        X = P; /*Calcula Fatorial de P em FAT2*/
        if (X == 0)
            FAT2 = 1;
        else {FAT2 = X;
            while (X > 1)
            {
                FAT2 =FAT2 * (X-1);
                X--;
            }
        }

        X = N-P; /*Calcula Fatorial de (N-P) em FAT3*/
        if (X == 0)
            FAT3 = 1;
        else {FAT3 = X;
            while (X > 1)
            {
                FAT3 =FAT3* (X-1);
                X--;
            }
        }

        COMB = (FAT1)/(FAT2 * FAT3);
        printf(“Combinacao = %i \n “,COMB);
    }
}
```

6.1. Funções : Introdução

- Na linguagem C as funções oferecem um meio de decompor repetidamente um problema em problemas menores.
- Um programa grande pode ser escrito como uma coleção de funções onde cada uma executa uma tarefa específica.

Estrutura Geral de um Programa em C

Diretivas de Pré-processamento

```
#include ....
```

```
#define ....
```

Declarações de Variáveis Globais /* var que podem ser acessadas em qq pt do prog*/

Protótipos Globais das Funções /*tipo de retorno e tipo dos parâmetros*/

```
void main(void ) /* função principal –início da execução do programa*/
```

```
{
```

```
  declarações de variáveis locais; /* var que só podem ser acessadas dentro do main*/
```

```
  comandos; /*incluindo as chamadas às funções*/
```

```
  ....
```

```
}
```

```
Tipo func1 (declaração de parâmetros) /* definição da função1 */
```

```
{
```

```
  declarações de variáveis locais; /* var que só podem ser acessadas dentro de func1*/
```

```
  comandos;
```

```
}
```

```
:
```

```
:
```

```
Tipo funcN (declaração de parâmetros) /* definição da funçãoN */
```

```
{
```

```
  declarações de variáveis locais; /* var que só podem ser acessadas dentro de funcN*/
```

```
  comandos;
```

```
}
```

6.2. Funções : Três Etapas Básicas

O uso de funções em C depende de três etapas básicas:

Protótipo, Chamada e Definição.

- Todas as funções precisam de um **protótipo**
Funções de biblioteca: constam em `<XXX.h>`
Nossas Funções: especificado após os include
- O **início da execução** da função acontece quando a função é **chamada** em algum ponto do programa.
- A partir da chamada a execução é transferida para a **DEFINIÇÃO** da função. A função apenas irá ser executada corretamente se todos os comandos necessários para isto estiverem presentes na sua definição.

```
#include <stdio.h> /* Protótipo da função printf */
#include <math.h> /* Protótipo da função pow*/
```

```
void test_num(int); /* Protótipo da test_num */
```

```
void main (void) /* definição da função main */
```

```
{
```

```
    int I=12;
```

```
    float x = -2.4f;
```

```
    printf(“%f\n”, x); /* chamada a printf*/
```

```
    printf(“%d | %f | %f\n”, I, pow(I,2), pow(I,3) ); /* chamada a printf*/
```

```
    test_num(I); /* chamada a test_num*/
```

```
}
```

```
void test_num(int X) /* definição da função test_num*/
```

```
{
```

```
    if (X%2==0) printf(“%i é par”, X); /* chamada a printf*/
```

```
    else printf(“%i é impar”, X); /* chamada a printf*/
```

```
}
```

6.2.1 Protótipo das funções (declaração)

tipo **nome_da_função(tipo dos argumentos);**
 ↳ básicos ↳ mesma regra
 + para nome
 modificadores das variáveis
 ou **void**
 void ou
 lista dos tipos de todos
 os argumentos recebidos pela função

Ex: **void menu(void); void test_num(int); long int fatorial(int);**

6.2.2 Chamada das funções

nome_da_função(lista dos argumentos);

Exemplos: **menu(); testa_num(6);**
 clrscr(); FAT = fatorial(10);

6.2.3 Definição das funções

tipo **nome_da_função(declaração dos parâmetros)**
 {
 Declaração das variáveis locais ↓
 Comandos; declaração de todos os
 argumentos recebidos pela função
 } /*Na definição da função, os argumentos recebidos são armazenados
 em variáveis denominadas de parâmetros*/

Exemplo: **void test_num(int X)**
 {
 if (X%2==0) printf(“%i é par”, X);
 else printf(“%i é impar”, X);
 }

6.3. Funções de biblioteca

As funções de biblioteca mais comuns são:

printf usada para imprimir valores na tela

protótipo: consta no arquivo `<stdio.h>`

chamada: a chamada à função **printf** pode ocorrer no main ou em qualquer outra função do programa Ex. `printf(“%f”,y);`

definição: a função **printf** já vem pronta para ser utilizada (não temos acesso ao código fonte desta função). Portanto, não temos acesso à definição dela.

scanf usada para entrada de dados pelo usuário

protótipo: consta no arquivo `<stdio.h>`

chamada: a chamada à função **scanf** pode ocorrer no main ou em qualquer outra função do programa. Ex. `scanf(“%i”&a);`

definição: a função **scanf** já vem pronta para ser utilizada (não temos acesso ao código fonte desta função). Portanto, não temos acesso à definição dela.

pow usada para o cálculo de potência x^y

protótipo: consta no arquivo `<math.h>`

chamada: a chamada à função **pow** pode ocorrer no main ou em qualquer outra função do programa Ex. `pow(10,3);`

definição: a função **pow** já vem pronta para ser utilizada (não temos acesso ao código fonte desta função). Portanto, não temos acesso à definição dela.

clrscr usada para limpar a tela

protótipo: consta no arquivo `<conio.h>`

chamada: a chamada à função **clrscr** pode ocorrer no main ou em qualquer outra função do programa Ex. `clrscr();`

definição: a função **clrscr** já vem pronta para ser utilizada (não temos acesso ao código fonte desta função). Portanto, não temos acesso à definição dela.

Para as nossas funções, precisamos incluir também os comandos necessários para que a tarefa seja executada, isto é, incluir a **definição**

Exemplo de Funções em C

Declaração ou **Protótipo**, **Chamada**, e **Definição** de Funções

```
#include <stdio.h> /* Protótipo da função printf */

void imp_mens1(void);      /* Protótipo da função imp_mesn1 */
void imp_mens2(int);      /* Protótipo da função imp_mens2 */
void imp_mens3(int, float); /* Protótipo da função imp_mens3 */

void main (void)          /* definição da função main */
{
    int I;
    float x = -2.4f;

    I = 12;
    imp_mens1( ); /*chamada da função imp_mesn1*/
    imp_mens2(I); /*chamada da função imp_mens2*/
    imp_mens3(I,x); /*chamada da função imp_mens3*/
}

void imp_mens1 (void) /*definição da função imp_mesn1*/
{
    printf(" O valor da constante e %d \n", 10); /*chamada printf*/
}

void imp_mens2 (int var) /*definição da função imp_mens2*/
{
    printf(" O valor da var recebida c/ argumento e %d \n", var); /*chamada printf*/
    printf(" O valor da constante e %d \n", 10); /*chamada printf*/
}

void imp_mens3 (int var1, float var2) /*definição da função imp_mens2*/
{
    printf(" Os valores das var recebidas como arg sao %d e %.2f\n", var1,var2);
    printf(" O valor da constante e %d \n", 10); /*chamada printf*/
}
```

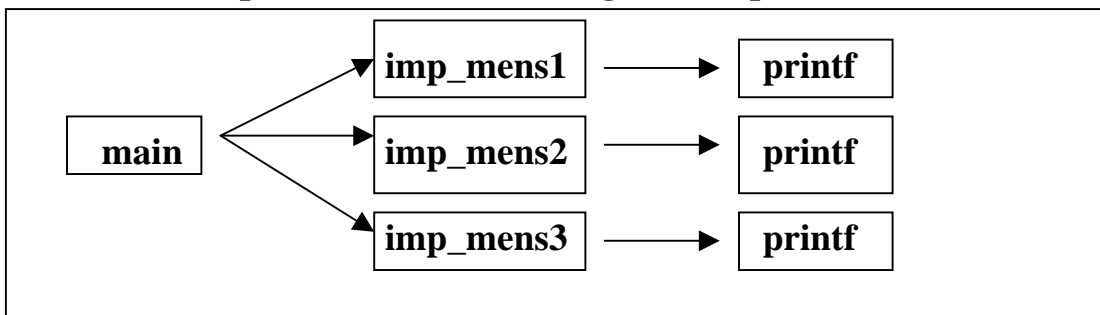
6.4 CHAMADA de Funções: ONDE serão executadas

A função somente é executada no ponto onde ocorre a **CHAMADA**. Se **não há nenhuma chamada**, a função **NÃO É EXECUTADA**.

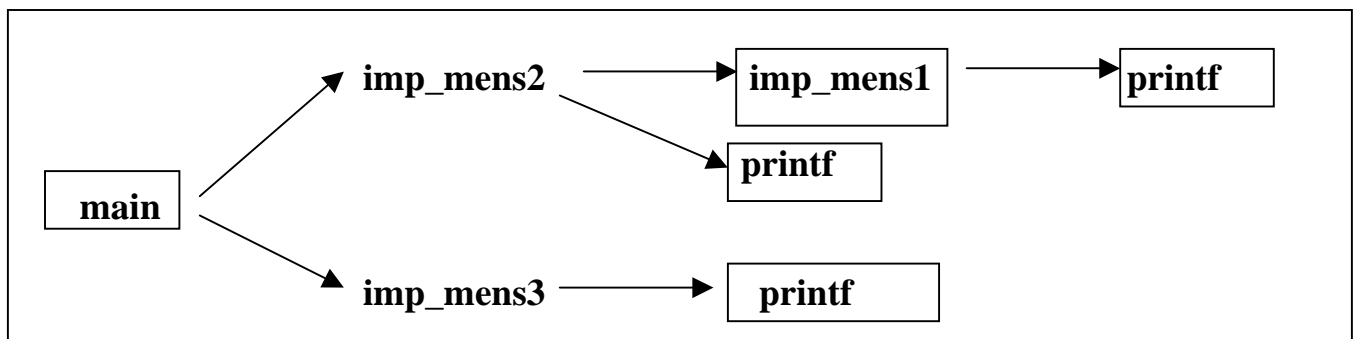
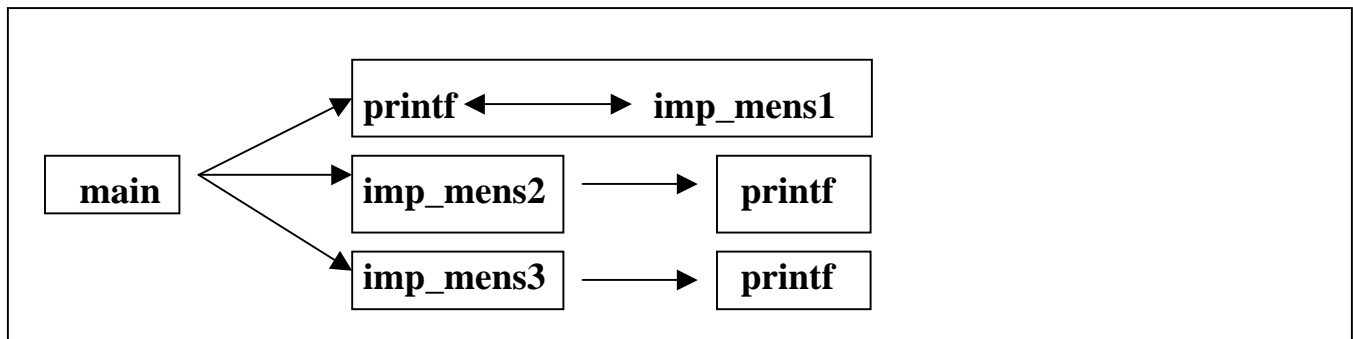
A função main é chamada pelo compilador e é um exemplo de função que chama outras no programa. Na função main temos

- as chamadas às funções que nós mesmos implementamos,
- as chamadas às funções de biblioteca (printf, scanf, pow)

No exemplo anterior temos o seguinte esquema de chamadas de funções



Refaça o programa anterior para que as chamadas das funções siga as estruturas mostradas a seguir e o resultado seja o mesmo do programa original



6.4.1 Argumentos (na chamada) das funções

`printf` é um exemplo de função bastante utilizado em C. Quando utilizamos a função `printf`, sempre fornecemos um ou mais valores para a função.

A função [printf](#) é uma função de biblioteca (pré-definida)

Neste caso temos apenas

protótipo que consta no arquivo `stdio.h`

chamada que pode ocorrer em qualquer ponto do programa e marca o início da execução da função

- Aquilo que fornecemos para a função é denominado argumento – na chamada vem sempre após o nome da função e **entre parênteses**.

Exemplo:

```
#include <stdio.h> /* este arquivo contém o protótipo da função printf*/

void main(void)
{
    char L='A';
    float x=2.5f;
    int A=2,B=3;
    printf( "imprimindo %d, %c, %f e %i", 10, L, x, A+B );
}
```

CHAMADA

- O **primeiro argumento** é a string "imprimindo %d, %c, %f e %i" que determina a formatação do que será impresso, ou seja, como a lista de parâmetros será impressa
- O **segundo argumento** é uma **constante inteira**, cujo valor 10 é passado para a função
- O **terceiro argumento** é uma **variável do tipo char**, cujo valor 'A' é passado para a função
- O **quarto argumento** é uma **variável do tipo float**, cujo valor 2.5 é passado para a função
- O **quinto argumento** é uma **expressão**, cujo valor 5 é passado para a função

6.5 DEFINIÇÃO de Funções: Como são executadas

Na **DEFINIÇÃO** devem constar

- Declaração do **tipo da função**: void, float, char, int, etc...
- As declarações dos **parâmetros**: após o nome da função e entre parênteses
- Os **comandos** necessários a execução da função

Exemplo:

```
#include <stdio.h> /* este arquivo contém o protótipo da função printf*/
void test_num(int); /* protótipo da função test_num*/
```

```
void main(void)
{
    char L='A';
    float x=2.5f;
    int A=2,B=3;

    printf( "imprimindo %d, %c, %f e %i", 10, L, x, A+B );
    test_num(A);
    test_num(B);
}
```

```
void test_num(int X)
{
    if (X%2==0) printf(“%i é par”, X);
    else printf(“%i é impar”, X);
}
```

Função test_num

Tipo da função: void

Declaração dos parâmetros: int X

Comandos: seleção composta (if-else) e duas chamadas a printf

6.6. Funções Retornando Resultados

6.6.1 Funções de Biblioteca: Já vimos exemplos de funções da biblioteca matemática que retornam resultados. Exemplo: a função `pow(X,N)` que devolve o valor de X^N

1. Os resultados retornados pelas funções podem ser armazenados em outras variáveis. Exemplo: `A = pow(Y,2);`
2. Utilizados diretamente em expressões. Exemplo: `B = pow(Y,2)+pow(Z,3);`
3. Ou utilizados na chamada a outras funções. Exemplo: `printf(“%f”,pow(X,4));`

6.6.2 Funções Definidas pelo Programador

- Como elaborar as nossas próprias funções para que retornem um resultado para outras funções que a chamam?
 - a) **Declarar o tipo** retornado pela função (no protótipo e na definição)
 - b) **Retornar o resultado** através do comando **RETURN**.
 - c) **Armazenar/utilizar** o resultado sob uma 3 formas detalhadas em 5.6.1

Exemplo:

```
#include <stdio.h> /* este arquivo contém o protótipo da função printf*/
int MDC(int,int); /* protótipo da função MDC*/

void main(void)
{
    int v1, v2, MaxDivCom;
    scanf(“%i %i”,&v1,&v2);
    MaxDivCom=MDC(v1,v2);
    printf(“O maximo divisor comum entre %i e %i e’ %i, v1,v2,MaxDivCom);
}

int MDC(int u, int v) /*início da definição da função MDC*/
{
    int temp;
    while(v!=0)
    {
        temp = u%v;
        u =v;
        v = temp;
    }
    return(u); /* Este comando return(u) equivale a fazer MCD = u */
}
```

6.7 Funções: Escopo de Variáveis

Variáveis Globais: São variáveis definidas fora de qualquer função. Todas as funções podem acessar ou alterar o seu valor.

Variáveis Locais: São variáveis definidas dentro do escopo de uma função (dentro do corpo). Somente a função pode acessar a variável ou alterar o seu valor.

<pre>#include<stdio.h> #include<math.h> void m1(void); /*protótipo da função m1*/ void m2(void); /*protótipo da função m2*/ int X; /* declaração da variável global X*/</pre>	<p>Escopo das Variáveis</p> <p>X</p>
<pre>void main(void) { int I; float RAIZ; scanf("%d",&I); printf("O valor lido foi %d \n",I); X=2; RAIZ=pow(X,1.0/2); printf("A raiz quadrada de %d é %f\n",X,RAIZ); m1(); m2(); printf(" X=%d, RAIZ=%f \n",X,RAIZ); }</pre>	<pre>main { I RAIZ }</pre>
<pre>void m1(void) { printf("O valor da variável global X era %d \n",X); X = 3; printf("O valor da variável global X foi alterado para %d \n",X); }</pre>	<pre>m1 { m1 não tem var local }</pre>
<pre>void m2(void) { int A=10; float RAIZ; printf("O valor da variável local A é %d \n",A); RAIZ = pow(X,1.0/2); printf("A raiz quadrada de %d é %f\n",X,RAIZ); }</pre>	<pre>m2 { A RAIZ }</pre>

6.7.1 Uso de Variáveis Globais:

O uso de variáveis globais acontece quando as funções do programa não recebem nenhum argumento ou não retornam nenhum valor (tipo de retorno = void)

```
/* Exemplo de Programa que calcula o fatorial de um número utilizando variáveis GLOBAIS */
```

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

```
int N; /* declaracao da variavel global N*/
```

```
long int Fat; /* declaracao da variavel global Fat*/
```

```
void fatorial(void); /* prototipo da funcao fatorial (não retorna valor e não recebe argumento) */
```

```
void main(void)
{
    printf("Entre com um valor inteiro: ");
    scanf("%d",&N);
    fatorial( );
}
```

```
void fatorial(void)
{
    int Numero=N;
    if(Numero>=0)
    {
        if(Numero==0)
            Fat = 1; /* definição de fatorial de 0*/
        else
        {
            Fat = Numero;
            while(Numero > 1)
            {
                Fat = Fat*(Numero-1);
                Numero--;
            }
        }
        printf("O fatorial de %d e igual a %ld \n",N,Fat);
    }
    else printf("Não existe fatorial de %d (valor negativo) \n",N);
}
```

6.7.2 Uso de Variáveis Locais:

O uso de variáveis locais acontece quando as funções do programa recebem argumentos e/ou **retornam** algum valor (trocam informações)

```
/* Exemplo de Programa que calcula o fatorial de um número utilizando somente
variáveis LOCAIS */
#include<stdio.h>
```

```
long int fatorial(int); /* prototipo da funcao fatorial (retorna valor
e recebe argumento) */
```

```
void main(void)
{
    int N;          /* declaracao da variavel local N*/
    long int F;     /* declaracao da variavel local F */
    printf("Entre com um valor inteiro: ");
    scanf("%d",&N);
    F = fatorial(N);
    if (F != -1) printf("O fatorial de %d e' %ld \n",N,F);
    else printf("Não existe fatorial de numero negativo \n");
}
```

```
long int fatorial(int N)
{
    long int Fat;
    if(N>=0)
    {
        if(N==0 )
            Fat = 1; /* definição de fatorial de 0*/
        else
        {
            Fat = N;
            while(N > 1)
            {
                Fat = Fat*(N-1);
                N--;
            }
            return(Fat); ←—————
        }
    }
    else return(-1); ←—————
}
```

Exemplos de Programas com Funções (Parte I)

```

/*****
Exemplo I-A: função que recebe argumentos e retorna
Valor (utilização de variáveis LOCAIS)
tipo_retornado : char
tipo dos parâmetros : int
*****/
#include <stdio.h>

```

```
char testa_sinal(int );
```

```

void main(void)
{
    int i; /* variáveis locais de main */
    char resp;
    do
    {
        printf("Entre com um valor inteiro positivo \n");
        scanf("%d",&i);
        resp = testa_sinal(i);
    }while(resp!='n');
}

```

```

char testa_sinal(int a)
{
    char sinal; /* variável local cujo valor será retornado a função main */

    if(a>=0)
    {
        sinal = 'p';
        printf("Valor OK \n");
    }
    else
    {
        sinal = 'n';
        printf("Valor nao permitido\n");
    }
    return(sinal);
}

```

<pre> /***** Exemplo I-B: função que recebe argumentos e retorna modulo do argumento tipo_retorna : float tipo dos parâmetros : float *****/ </pre>
<pre> include <stdio.h> float modulo(float); </pre>
<pre> void main(void) { float f1,f2,res; int i1; f1 = -12.4f; f2 = -4.2f; i1 = -3; res = modulo(f1); printf("f1 = %2.2f, f1 = %2.2f\n",f1,res); res = modulo(f1) + modulo(f2); printf("A soma de f1 + f2 e' %2.2f\n",res); res = modulo((float) i1); printf("Cast de argumento inteiro e' %2.2f\n",res); res = modulo(i1); printf("Conversao de argumento inteiro , %2.2f\n",res); printf("divisao por inteiro : %2.2f\n",modulo(-6.0)/4); } </pre>
<pre> float modulo(float x) { if(x<0) x = -x; return(x); } </pre>

Exemplo I-C: Geral

```
#include <stdio.h> /* contem protótipos e definições básicas de entrada/saída */
#include <conio.h> /* contem protótipo da função getch */
```

```
int func1(void); /*protótipos globais das funções definidas no programa */
void func2(int);
float y; /* variavel global */
```

```
void main(void)
```

```
{
    int p,var;
    printf("\n*****Funcao main*****\n");
    printf("Entre com um valor inteiro : \n");
    scanf("%d",&p);
    printf("variavel local p da funcao main = %d\n",p);
    y=12.5f;
    printf("Variavel global y = %3.3f\n",y);
    printf("Transfere comando para func1 \n");
    var = func1( );
    printf("\nNovamente funcao main \n");
    printf("retorno da funcao func1 e' %d\n",var);
    printf("Termina a execucao\n");
    getch( ); /* espera qualquer valor lido do teclado */
}
```

```
int func1(void)
```

```
{
    int f1,arg=2,retorno_func1; /* variáveis locais de func1 */
    printf("*****Funcao func1*****\n");
    printf("Entre com um numero\n");
    scanf("%d",&f1);
    printf("variavel local f1 = %d\n",f1);
    printf("variavel local arg = %d passada como argumento \
para a funcao func2\n",arg);
    printf("trasfere comando para func2\n");
    func2(arg);
    printf("\nNovamente funcao func1\n");
    printf("variavel global y = %3.3f\n",y);
    retorno_func1 = f1;
    printf("retorna para main o valor de f1 = %d\n",f1);
    return(retorno_func1);
}
```

```

void func2(int parm_x)
{
    int f2=4;
    printf("*****Funcao func2*****\n");
    printf("variavel local f2 = %d \n",f2);
    printf("variavel global y = %3.3f \n",y);
    printf("recebeu de func1 como argumento, o valor %d \n",parm_x);
}

```

Entradas via teclado :

100

200

Saída do Programa :

*****Funcao main*****

Entre com um valor inteiro :

variavel local p = 100

Variavel global y = 12.500

Transfere comando para func1

*****Funcao func1*****

Entre com um numero

variavel local f1 = 200

variavel local arg = 2 passada como argumento para a funcao func2

trasfere comando para func2

*****Funcao func2*****

variavel local f2 = 4

variavel global y = 12.500

recebeu de func1 como argumento, o valor 2

Novamente funcao func1

variavel global y = 12.500

retorna para main o valor de f1 = 200

Novamente funcao main

retorno da funcao func1 e' 200

Termina a execução

6.8. Funções: Passagem por Valor X Passagem por Referência

a)Transferência de Argumentos por Valor (passagem por valor): Neste caso, **apenas o valor da variável é passado na chamada da função**. As alterações feitas nos parâmetros não estão acessíveis fora do corpo da função.

Exemplo:

```
int N=2;
float P=7.5f;

printf(“%d %.2f”, N,P);
```

Neste caso apenas os valores das variáveis N e P são passados para a função printf

b)Transferência de Argumentos por Referência (passagem por referência): Neste caso o **endereço da variável é passado na chamada da função**. Com isso, a função pode acessar a área de memória da variável, alterar o seu conteúdo e as alterações passam a ser visíveis para todas as outras funções que utilizam estas variáveis.

Exemplo:

```
int N;
float P;

scanf(“%d %f”, &N,&P);
```

Neste caso os endereços das variáveis N e P são passados para a função scanf que pode alterar o conteúdo destas variáveis, e outras funções (como a main por exemplo) conseguem perceber estas alterações.

6.8.1 Passagem por Valor: Protótipo, Definição e Chamada

Na passagem por valor, os protótipos, definição e chamada das funções acontecem como os exemplos mostrados anteriormente (a função só pode alterar localmente os conteúdos das variáveis):

Protótipo das funções

tipo nome_da_função(tipo dos parâmetros);

Definição das funções

tipo nome_da_função(declaração dos parâmetros);
{
 Declaração das variáveis locais
 Comandos;
}

Chamada das funções

nome_da_função(lista dos argumentos);

Exemplo

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

```
void func_ex_chamada_por_valor(int, float );
```

← Protótipo

```
void main(void)
{
    int a=2; float y=3.5f;    /* variáveis locais de main */
    func_ex_chamada_por_valor(a,y);
    printf(“%d %f \n”, a, y); /* imprime 2 e 3.500000*/
}
```

← Chamada

```
void func_ex_chamada_por_valor(int a, float y)
{
    int c=3; float z=4.6f;
    a = a+2;
    y = y+2.5f;
    printf(“Variaveis locais: c=%d, z=%f \n”,c,z);
    printf(“Variáveis cujos valores foram recebidos de main: \
e foram alterados localmente a=%d, y=%f \n”,a,y);
}
```

← Definição

6.8.2 Passagem por Referência: Protótipo, Definição e Chamada

Na passagem por referência, os protótipos, definição e chamada das funções são alterados para troca de endereços das variáveis ao invés dos valores das variáveis (a função pode alterar globalmente o conteúdo das variáveis):

Protótipo das funções

tipo nome_da_função(tipo dos parâmetros seguidos por *);

Definição das funções

tipo nome_da_função(declaração dos parâmetros -ponteiros);

{

Declaração das variáveis locais

Comandos; /* todo acesso ao conteúdo da variável ponteiro deve ser feito acrescentando-se o '*' */

}

Chamada das funções

nome_da_função(lista dos endereços dos argumentos);

Exemplo

```
#include <stdio.h>
void func_ex_chamada_por_refer(int *, float *); ← Protótipo

void main(void)
{
    int a=2; float y=3.5f;    /* variáveis locais de main */
    func_ex_chamada_por_refer(&a, &y); ← Chamada
    printf(“%d %f \n”, a, y); /* imprime 4 e 6.000000*/
}

void func_ex_chamada_por_refer(int *a, float *y) ← Definição
{
    int c=3; float z=4.6f;
    *a = *a+2; → Acesso ao conteúdo
    *y = *y+2.5f; → Acesso ao conteúdo

    printf(“Variaveis locais: c=%d, z=%f \n”, c, z);
    printf(“Variáveis cujos valores foram recebidos de main: \
e foram alterados globalmente a=%d, y=%f \n”, *a, *y);
}

```

Exemplos de Programas com Funções (Parte II)

Nos Exemplos a seguir todas as variáveis declaradas são do tipo LOCAL

E entre as funções existe passagem por valor e/ou referência

```
/******
```

```
Exemplo II-A: funcao que recebe argumentos (A,B, e C)
e os endereços dos argumentos X e Y que guardarão as raízes da equação
tipo_retornado : int (flag que indica se existem raízes reais)
tipo dos parâmetros : valores e endereços *****/
#include <stdio.h>
#include <math.h>
```

```
int calcula_raiz(float, float, float, float *, float * );
```

```
void main(void)
{
    int resp; char continua;          /* variáveis locais de main */
    float A,B,C,X1,X2;               /* variáveis locais de main */
    do
    {
        printf("Entre com os valores A, B e C da eq. do seg. grau\n");
        scanf("%f %f %f", &A, &B, &C);
        resp = calcula_raiz(A,B,C,&X1,&X2);
        if (resp != -1)
            printf("As raízes são X1 = %.2f e X2 = %.2f \n",X1,X2);
        else
            printf("Não existem raízes reais \n");
        printf("Digite (n) ou (N) para parar e qq. Letra para continuar\n");
        scanf("\n%c",&continua);
    }while(continua!='n' && continua!= 'N' );
}
```

```
int calcula_raiz(float A, float B, float C, float *X1, float *X2)
```

```
{
    float delta;
    delta = B*B - 4*A*C;
    if (delta < 0)
        return(-1); /* não existem raízes reais */
    else
    {
        *X1 = -B + sqrt(delta)/2*A; /* sqrt - calc. Raiz quadrada*/
        *X2 = -B - sqrt(delta)/2*A;
        return(1);
    }
}
```

```
/******
```

Exemplo II-B: funcao que recebe como argumentos os endereços de duas variáveis e troca os seus valores

tipo_retornado : void

tipo dos parâmetros : endereços das variáveis *****/

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

```
void troca( float *, float * );
```

```
void main(void)
{
    char continua;          /* variável local de main */
    float A,B,C;           /* variáveis locais de main */
    do
    {
        printf("Entre com os valores A, B e C a serem ordenados\n");
        scanf("%f %f %f", &A, &B, &C);

        /* Armazena menor em A se necessário*/
        if (A > C && B > C)
            troca(&A,&C);
        else if(A > B && B < C)
            troca(&A,&B);

        /* Armazena intermediário B se necessário */
        if (B >C)
            troca(&B,&C);
        printf("Os valores ordenados são %.2f %.2f %.2f\n",A,B,C);
        printf("Digite (n) ou (N) para parar e qq. Letra para continuar\n");

        scanf("\n%c",&continua);
    }while(continua!='n' && continua!= 'N' );
}
```

```
void troca(float *X, float *Y)
{
    float auxiliar;

    auxiliar = *X;
    *X = *Y;
    *Y = auxiliar;
}
```

7 Funções e Estruturas de Dados

7.1 Funções e Vetores

Vetores como Variáveis Globais: Neste caso, os vetores são definidos fora de qualquer função. Todas as funções podem acessar ou alterar o valor de cada elemento.

Vetores como Variáveis Locais: Neste caso os vetores são definidos dentro das funções e, como no caso das variáveis simples, a passagem pode ser por valor ou por referência.

<pre>#include<stdio.h> void m1(void); /*protótipo da função m1*/ void m2(void); /*protótipo da função m2*/ int X[5]; /* declaração do vetor X como var. global*/</pre>	<p>Escopo das Variáveis</p> <p>X[0]...X[4]</p>
<pre>void main(void) { int I,Vet[5]; for (I=0;I<5;I++) { scanf("%d",&Vet[I]); X[I] = 0; } m1(); m2(); for (I=0;I<5;I++) printf("\n Vet[%d]=%d X[%d]= %d \n",I,Vet[I],I,X[I]); }</pre>	<pre>main { I Vet[0]..Vet[4] }</pre>
<pre>void m1(void) { int I; for (I=0;I<5;I++) X[I] = 2*I; }</pre>	<pre>m1 { I }</pre>
<pre>void m2(void) { int A,Vet[5]={1,2,3,4,5}; for (A=0;A<5;A++) printf("%d ",Vet[A]); }</pre>	<pre>m2 { A Vet[0] ... Vet[4] }</pre>

7.1.1 Funções e Vetores: Passagem do Elemento do Vetor

Quando um elemento do Vetor é passado como parâmetro, a passagem segue as regras de variáveis simples.

a) Transferência de Elementos por Valor:

```
int N;
float P[10]={2,4,6,8,10,12,14,16,18,20};
for (N=0;N<10;N++)
    printf("%.2f ", P[N]);
```

Neste caso o valor de cada elemento P[N] do vetor P é passado para printf

```
#include<stdio.h> /* Programa exemplo de passagem por valor*/
int testa_par_acima10_elem(int); /*protótipo da função*/

void main(void)
{
    int I,resp,cont_par=0, cont_acima10=0,Vet[50];
    for (I=0;I<50;I++)
    {
        scanf("%d",&Vet[I]);
        resp = testa_par_acima10_elem(Vet[I]); /*passagem por valor */
        switch(resp)
        {
            case 1: cont_acima10++; break;
            case 2: cont_par++; break;
            case 3: cont_acima10++; cont_par++; break;
        }
    }
    for (I=0;I<50;I++)
        printf("Vet[%d]=%d \n",I,Vet[I]);
    printf("Deste vetor %d elemento(s) são par e %d estão acima de 10
\n",cont_par,cont_acima10);
}

int testa_par_acima10_elem(int a) /* o valor do elemento é recebido */
{
    if (a%2==0)
    {
        if(a>10) return(3); /*se é par e acima de 10 retorna flag 3*/
        else return(2); /* se é par retorna flag 2*/
    }
    else if(a>10) return(1); /* se é maior do que 10 retorna flag 1*/
    else return(0); /* caso contrario retorna flat 0*/
}
```

b) Transferência de Elementos por Referência:

```
int N;
float P[10];
for (N=0;N<10;N++)
    scanf("%f", &P[N]);
```

Neste caso o endereço de cada elemento P[N] do vetor P é passado para scanf

```
#include<stdio.h> /*Programa Exemplo de Passagem por referencia*/
int soma_2_par_acima10_elem(int *); /*protótipo da função*/

void main(void)
{
    int I,resp,cont_par=0, cont_acima10=0,Vet[50];
    for (I=0;I<50;I++)
    {
        scanf("%d",&Vet[I]);
        resp = soma_2_par_acima10_elem(&Vet[I]); /*passagem por referência*/
        switch(resp)
        {
            case 1: cont_acima10++; break;
            case 2: cont_par++; break;
            case 3: cont_acima10++; cont_par++; break;
        }
    }
    for (I=0;I<50;I++)
        printf("Vet[%d]=%d \n",I,Vet[I]);
    printf("Deste vetor %d elemento(s) são par e %d estão acima de 10
\n",cont_par,cont_acima10);
}

int soma_2_par_acima10_elem(int *a) /* o endereço do elemento é recebido */
{
    if (*a%2==0)
    {
        *a=*a+2; /*se é par soma 2*/
        if(*a>10) return(3); /*se é par e acima de 10 retorna flag 3*/
        else return(2); /* se par retorna flag 2*/
    }
    else if(*a>10) return(1); /* se é maior do que 10 retorna flag 1*/
    else return(0); /* caso contrario retorna flat 0*/
}
```

7.1.2 Funções e Vetores: Passagem do Vetor Completo

Quando o nome do Vetor é passado como parâmetro, a passagem é sempre por referência, pois a função tem acesso ao endereço do vetor e pode alterar todos os valores, tornando a alteração visível para todas as outras funções.

```
char cidade[10];
scanf("%s",cidade);
```

Neste caso o endereço do vetor `cidade` é passado para `scanf` que pode alterar cada elemento, tornando a alteração visível para toda a função que a utiliza.

Exemplo:

```
#include<stdio.h>
void lê_soma_2_par_acima10_vet(int [ ],int *, int *); /*protótipo da função*/

void main(void)
{
    int I,resp,cont_par=0, cont_acima10=0,Vet[50];
    le_soma_2_par_acima10_vet(Vet,&cont_par,&cont_acima10);
    for (I=0;I<50;I++)
        printf("Vet[%d]=%d \n",I,Vet[I]);
    printf("Deste vetor %d elemento(s) são par e %d estão acima de 10
\n",cont_par,cont_acima10);
}

void lê_soma_2_par_acima10_vet(int Vet[],int *cont_par, int *cont_acima10)
{
    int I;
    for (I=0;I<50;I++)
    {
        scanf("%d",&Vet[I]);
        if (Vet[I]%2==0)
        {
            Vet[I] =Vet[I]+2;
            *cont_par= *cont_par+1;
        }
        if(Vet[I]>10)
            *cont_acima10= *cont_acima10+1;
    }
}
```

7.2. Funções e Matrizes

Matrizes como Variáveis Globais: As matrizes são definidas fora de qualquer função. Todas as funções podem acessar ou alterar o valor de cada elemento da matriz.

Matrizes como Variáveis Locais: Neste caso as matrizes são definidas dentro das funções e, como no caso das variáveis simples, a passagem pode ser por valor ou por referência.

<pre>#include<stdio.h> void m1(void); /*protótipo da função m1*/ void m2(void); /*protótipo da função m2*/ int M[2][5]; /* declaração da matriz M como var. global*/</pre>	<p>Escopo das Variáveis</p> <p>M[0][0]...X[1][4]</p>
<pre>void main(void) { int L,I,Mat[2][5]; for(L=0;L<2;L++) for (I=0;I<5;I++) { scanf(“%d”,&Mat[L][I]); M[L][I] = 0; } m1(); m2(); for(L=0;L<2;L++) for (I=0;I<5;I++) printf(“Mat[%d][%d]=%d M[%d][%d]= %d \n”, L, I, Mat[L][I], L, I, M[L][I]); }</pre>	<pre>main { L I Mat[0][0]..Mat[1][4] }</pre>
<pre>void m1(void) { int L,I; for(L=0;L<2;L++) for (I=0;I<5;I++) M[L][I] = 2*(L+I); }</pre>	<pre>m1 { I }</pre>
<pre>void m2(void) { int a,A,Mat[2][5]={ {0,0,0,0,0}, {1,2,3,4,5} }; for(a=0;a<2;a++) for (A=0;A<5;A++) printf(“%d ”,Mat[a][A]); }</pre>	<pre>m2 { a A Mat[0][0]..Mat[1][4] }</pre>

7.2.1 Funções e Matrizes: Passagem do Elemento da Matriz

Quando um elemento da Matriz é passado como parâmetro, a passagem segue as regras de variáveis simples.

a) Transferência de Elementos por Valor:

```
int N,M;
float P[3][4]={ {2,4,6,8}, {10,12,14,16}, {18,20,22,24} };
for (N=0;N<3;N++)
    for(M=0;M<4;M++)
        printf("%.2f ", P[N][M]);
```

O valor de cada elemento P[M][N] da matriz P é passado para printf

```
#include<stdio.h>
int testa_par_acima10_elem(int); /*protótipo da função*/
void main(void)
{
    int L,I,resp,cont_par=0, cont_acima10=0,Mat[10][50];
    for(L=0;L<10;L++)
        for (I=0;I<50;I++)
            {
                scanf("%d",&Mat[L][I]);
                resp = testa_par_acima10_elem(Mat[L][I]); /*passagem por valor */
                switch(resp)
                {
                    case 1: cont_acima10++; break;
                    case 2: cont_par++; break;
                    case 3: cont_acima10++; cont_par++; break; }
            }
    for(L=0;L<10;L++)
        for (I=0;I<50;I++)
            printf("Mat[%d][%d]=%d \n",L,I,Mat[L][I]);
    printf("%d elemento(s) são par e %d são > 10 \n",cont_par,cont_acima10);
}

int testa_par_acima10_elem(int a) /* o valor do elemento é recebido */
{
    if (a%2==0)
        if(a>10) return(3); /*se é par e acima de 10 retorna flag 3*/
        else return(2); /* se par retorna flag 2*/
    else if(a>10) return(1); /* se é maior do que 10 retorna flag 1*/
    else return(0); /* caso contrario retorna flat 0*/
}
```

b) Transferência de Elementos por Referência:

```

int N,M;
float P[3][4];
for (N=0;N<3;N++)
    for(M=0;M<4;M++)
        scanf("%f", &P[N][M]);

```

O endereço de cada elemento P[N][M] da matriz P é passado para scanf

```

#include<stdio.h>
int soma_2_par_acima10_elem(int *); /*protótipo da função*/

void main(void)
{
    int L,I,resp,cont_par=0, cont_acima10=0,Mat[10][50];
    for (L=0;L<10;L++)
        for (I=0;I<50;I++)
            {
                scanf("%d",&Mat[L][I]);
                resp = soma_2_par_acima10_elem(&Mat[L][I]); /*passagem por refer */
                switch(resp)
                {
                    case 1: cont_acima10++; break;
                    case 2: cont_par++; break;
                    case 3: cont_acima10++; cont_par++; break; }
            }
    for (L=0;L<10;L++)
        for (I=0;I<50;I++)
            printf("Mat[%d][%d]=%d \n",L,I,Mat[L][I]);
    printf("%d elemento(s) são par e %d são > 10 \n",cont_par,cont_acima10);
}

int soma_2_par_acima10_elem(int *a) /* o endereço do elemento é recebido */
{
    if (*a%2==0)
    {
        *a=*a+2; /*se é par soma 2*/
        if(*a>10) return(3); /*se é par e acima de 10 retorna flag 3*/
        else return(2); /* se par retorna flag 2*/
    }
    else if(*a>10) return(1); /* se é maior do que 10 retorna flag 1*/
    else return(0); /* caso contrario retorna flat 0*/
}

```

7.2.2 Funções e Matrizes: Passagem da Linha da Matriz

Quando uma das linhas (ou colunas) da matriz é passada como parâmetro, a passagem é sempre por referência, pois a função tem acesso ao endereço do vetor linha (ou coluna) e pode alterar todos os valores, tornando a alteração visível para todas as outras funções.

```
char cidades[5][10]; int I;
  for (I=0;I<5;I++)
    scanf("%s",cidades[I]);
```

Neste caso o endereço de cada linha (vetor) da matriz `cidades` é passado para `scanf` que pode alterar cada elemento (caractere), tornando a alteração visível para toda a função que a utilize.

```
#include<stdio.h>
void le_soma_2_par_acima10_vet(int [ ],int *, int *); /*protótipo da função*/
```

```
void main(void)
{
  int L,I,resp,cont_par=0, cont_acima10=0,Mat[10][50];
  for(L=0;L<10;L++)
    le_soma_2_par_acima10_vet(Mat[L],&cont_par,&cont_acima10);
  for(L=0;L<10;L++)
    for (I=0;I<50;I++)
      printf("Mat[%d][%d]=%d \n",L,I,Mat[L][I]);
  printf("%d elemento(s) são par e %d são > 10 \n",cont_par,cont_acima10);
}
```

```
void le_soma_2_par_acima10_vet(int Vet[],int *cont_par, int *cont_acima10)
{
  int I;
  for (I=0;I<50;I++)
  {
    scanf("%d",&Vet[I]);
    if (Vet[I]%2==0)
    {
      Vet[I] =Vet[I]+2;
      *cont_par= *cont_par+1;
    }
    if(Vet[I]>10)
      *cont_acima10= *cont_acima10+1;
  }
}
```

7.2.3 Funções e Matrizes: Passagem da Matriz Completa

Quando o nome da Matriz é passado como parâmetro, a passagem é sempre por referência, pois a função tem acesso ao endereço da matriz e pode alterar todos os valores, tornando a alteração visível para todas as outras funções.

Exemplo:

```
#include<stdio.h>

void le_soma_2_par_acima10_mat(int [][][50],int *, int *); /*protótipo da função*/

void main(void)
{
    int L,I,resp,cont_par=0, cont_acima10=0,Mat[10][50];

    le_soma_2_par_acima10_mat(Mat,&cont_par,&cont_acima10);
    for(L=0;L<10;L++)
        for (I=0;I<50;I++)
            printf("Mat[%d][%d]=%d \n",L,I,Mat[L][I]);
    printf("%d elemento(s) são par e %d são > 10 \n",cont_par,cont_acima10);
}

void le_soma_2_par_acima10_mat(int Mat[][50],int *cont_par, int *cont_acima10)
{
    int L,I;
    for(L=0;L<10;L++)
        for (I=0;I<50;I++)
            {
                scanf("%d",&Mat[L][I]);
                if (Mat[L][I]%2==0) /* se é par soma 2 e retorna flag 2*/
                {
                    Mat[L][I] =Mat[L][I]+2;
                    *cont_par= *cont_par+1;
                }
                if(Mat[L][I]>10)
                    *cont_acima10= *cont_acima10+1;
            }
}
}
```