Linguagem C99:
Aula 1 – Conceitos Básicos

Software Básico, turma B

Prof. Marcos Caetano – CIC/UnB

Material Base: Prof. Marcelo Ladeira

Sumário

- Histórico e Primeiros Passos
  - Introdução às Funções
  - Introdução Básica a Entrada e Saída
- Lógica de Programação em C
  - Tipos, Operadores e Expressões
  - Controle de Fluxo
- Palavras Reservadas

Histórico

- C foi criada nos anos 70 por Dennis Ritchie (C tradicional ou KR)
  - Computador Digital PDP-11 com S.O. UNIX
  - Derivada da linguagem B de Ken Thompson
    - B é derivada da linguagem BCPL de Martin Richards.
- Aplicação
  - Inicial: desenvolver software básico, independente do hardware.
    - Usuários especializados
      - Interessados em geração de código compacto e eficiente
      - Gerência de memória ao cargo do programador
  - Características atuais
    - Economia de expressões
    - Moderno controle de fluxo e estruturas de dados
    - Conjunto de operadores rico e poderoso
    - Linguagem de nível intermediário
    - Biblioteca das funções que são dependentes do hardware (tipo I/O)
- Versões
  - 1989: C padrão ANSI ou C89
  - 1999: C padrão ISO ou C99
  - GCC: padrão C89 mais extensões GNU.
    - Use –std=c99 para gerar código no padrão ISO

Características introduzidas pelo C99

- Funções em linha (in line)
- Remoção de restrições sobre a localização da declaração de variáveis
- Adição de vários tipos de dados novos, como long long int (64 bits), tipo booleano explicito *_Bool*, tipo _Complex para número complexo, e _Imaginary para parte imaginário de um complexo
- Comentários de uma linha iniciados por //
- Várias funções de biblioteca novas
- Várias bibliotecas novas, tais como stdint.h

Vantagem da arquitetura 64 bits

- **Aumento significativo na velocidade de processamento**

```
int a, b, c, d, e;
for (a=0; a<100; a++) {
    b = a;
    c = b;
    d = c;
    e = d;
}
```

Histórico
Domínio de Aplicação do C

- Linguagem de programação genérica para criação de:
  - sistemas operacionais (SOFTWARE BÁSICO)
  - gerenciadores de bancos de dados (SOFTWARE BÁSICO)
  - programas de comunicação (SOFTWARE BÁSICO)
  - processadores de texto
  - planilhas eletrônicas
  - programas de automação industrial
  - programas de projeto assistido por computador
  - programas para a solução de problemas em Engenharia, Física, Química e outras ciências
  - etc ...

Primeiros Passos
Seqüências de Escapes

```
/* Exemplo: KR01FRST.C
   C A LINGUAGEM DE PROGRAMACAO.
      Brain W. KERNIGHAN e Dennis M
      RITCHIE
   Bell Laboratories, New Jersey. Traducao:
      EDISA - Eletronica Digital S.A, pagina 20
      */

#include <stdio.h>

int main ()
{
     printf("primeiro programa\t");
              /* impressao em video com
                 seqüência de escape */
     return (0);
}
```

Seqüências de escape

\n	LF (nova linha)
\t	Tabulacao horizontal
\b	BS (retrocesso)
\r	CR (retorno de carro)
\f	FF (muda a pagina)
\a	BEL (sinal audível)
\\	Imprime a contrabarra
\'	Imprime o apostrofo
\?	Imprime interrogação
\0ddd	valor octal de caracter ASCII
\0xdd	valor hexadecimal de ASCII
\0	caracter nulo: fim de string

Primeiros Passos
Caracteres Trigrafos (ISO 646)

??=	é substituído por	#
??(	é substituído por	[
??<	é substituído por	{
??/	é substituído por	\
??)	é substituído por	]
??>	é substituído por	}
??'	é substituído por	^
??!	é substituído por	|
??-	é substituído por	~

Primeiros Passos
Introdução às Funções

- Uma função é um bloco de código de programa, carregado apenas uma vez, e que pode ser usado diversas vezes na execução.
  - O uso de funções:
    - economiza memória e aumenta a legibilidade do programa
    - melhora a estruturação: facilita a depuração e o reuso.
- Um programa em C consiste de várias funções colocadas juntas.

Introdução às Funções
Exemplo de Função Sem Argumento

```
#include <stdio.h>
int mensagem();
int mensagem () /* Função simples:
     só imprime Ola! */ {
   printf ("Ola! ");
   return(0);
}
int main (){
   mensagem();
   printf ("Eu estou vivo!\n");
   return (0);
}
```

- **main() é sempre a primeira função a ser executada.**
- **A função mensagem() foi definida antes de main()**
  - **não precisa ser declarada novamente em main ()**
- **printf() não precisa ser declarada? Onde foi declarada?**
- **return não precisa ser declarada? Qual o tipo do valor retornado?**
  - **Não! É um comando de atribuição e não uma função!**
  - **O valor retornado é do tipo da função onde é executado.**

Introdução às Funções
Funções com Argumentos

```
#include <stdio.h>
/* Calcula x ^ 2 */
int square (int x){
   printf ("O quadrado eh
   %d",(x*x));

   return(0);
}
int main (){
   int num;
   printf ("Entre com um
   numero: ");
   scanf ("%d", &num);
   printf ("\n\n");
   square (num);
   printf ("\nO numero eh %d",
   num);
   return (0);
}
```

- **Não existe a função quadrado ou potência em biblioteca!**
- **Forma geral de uma função**

  tipo_de_retorno nome_da_função (lista_de_argumentos)
  {
  código_da_função
  }
- **Assinatura (onde for chamada)**

  tipo_de_retorno nome_da_função (lista_de_argumentos);
- **Passagem de argumentos:**
  - **lista de parâmetros**
    - **passagem por valor**
    - **passagem por referência**
  - **variáveis globais**
    - **definidas fora de uma função**
- **Funções podem retornar valores via o comando return.**

Introdução às Funções
Passagem de Argumentos por Valor

```c
#include <stdio.h>

/* Multiplica 3 números */
int mult (float a, float b, float c){
   printf ("%f",a*b*c);
   return (0);
}
int main (){
   float x,y;
   x=23.5;
   y=12.9;
   mult (x,y,3.87);
   return (0);
}
```

- Argumentos passados, inclusive constantes, são copiados para a variável de entrada da função.
- Haverá conversão de tipo porque trata-se de multiplicação de **float** em função que retorna **int** ?

Introdução às Funções
Passagem de Argumentos por Referência

```
troca (a,b); /* chamada */
```

Errado

```
void troca (int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

```
troca (&a,&b); /* chamada */
```

Certo

```
void troca (int * ax, int * ay)
{
    int temp;
    temp = *ax;
    *ax  = *ay;
    *ay  = temp;
}
```

Introdução às Funções
Funções que Retornam Valores

```c
#include <stdio.h>
int prod (int x, int y){
    return (x*y);
}
int main () {
    int saida;
    saida=prod (12,7);
    printf ("A saida e: %d\n",saida);
    return (0);
}
```

```c
#include <stdio.h>
float prod (float x, float y){
    return (x*y);
}
int main (){
    float saida;
    saida=prod (45.2, 0.0067);
    printf ("A saida e: %f\n",saida);
    return (0);
}
```

Funções inline

- Indica ao compilador que é uma boa idéia expandir a função onde ela aparece
  - Em um arquivo .h

```
inline double dabs(double x) {
    return (x < 0.0 ? -x : x);
}
```

  - Chamando em uma unidade de compilação (.c)

```
extern inline double dabs(double x);
```

Lógica de Programação em C
Nome das Variáveis

- Identificadores (máximo 31 caracteres)

  <id> ::= [<letra> | _]

  <id> ::= <id> [<letra> | _ | <numero>]

  - O nome de uma variável não pode ser:
    - palavra reservada
    - nome de função de biblioteca ou do programador
- Dicas
  - Nao use nomes iniciados por _
    - são utilizados pelas bibliotecas do próprio C.
  - Use minúsculas para variáveis e maiúsculas para constantes.

Lógica de Programação em C
Tipos de Dados

- **Objeto de Dados Elementares**

char	(caracter)
int	(inteiro)
float	(real)
double	(real extendido)
void	(sem tipo)
bool	(tipo booleano)
complex	(número complexo)

- **Modificadores de Tipo**

  signed, unsigned, short e long

Biblioteca stdint.h

- Exatos n bits

int8_t
int16_t
int32_t
int64_t
uint8_t
uint16_t
uint32_t
uint64_t

- Ao menos n bits

int_least8_t
int_least16_t
int_least32_t
int_least64_t
uint_least8_t
uint_least16_t
uint_least32_t
uint_least64_t

Tipos de Dados
Modificadores e Valores Típicos

Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Inicio	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%i	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i	-32.768	32.767
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3.4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

Lógica de Programação em C
Declaração e Iniciação de Variáveis

- **Variáveis devem ser declaradas com tipo_da_variável lista_de_variáveis;**
  - **As variáveis da lista possuem o mesmo tipo e são separadas por vírgula.**
  - **int é o tipo de dado padrão em C.**
    - **int pode ser suprimido na declaração de inteiros com modificadores.**

      long a, b, c;

      Short d, e, f;
    - **Funções sem declaração de tipo são consideradas int.**

Declaração e Iniciação de Variáveis
Pontos Onde Declarar Variáveis

```
#include <stdio.h>
int contador;
int func1 (int j) {
      /* aqui viria o código da funcao ... */
}

int main() {
      char condicao;
      int i;
      for (i=0; i<100; i=i+1) {
            float f2;
            /* etc ... ... */
            func1(i);
      }
            /* etc ... */
      return (0);
}
```

- **Fora das funções do programa**
  - **Variáveis globais.**
    - **contador (int).**
  - **Vistas a partir da declaração.**
  - **Duas variáveis globais não podem ter o mesmo nome.**
- **No início de um bloco de código**
  - **Variáveis locais.**
    - **condicao (char) e i (int) são visíveis em main () e no bloco for.**
    - **f2 (float) é visível apenas no bloco for.**
  - **Vistas apenas no bloco onde foram declaradas.**
  - **Duas variáveis locais só podem ter o mesmo nome se forem declaradas em funções distintas.**
- **Lista de parâmetros de função**
  - **Variáveis locais à função.**
    - **j (int).**

Lógica de Programação em C
Declaração e Iniciação de Variáveis

- **Variáveis podem ser iniciadas na declaração com:**

  tipo_da_variável nome_da_variável = constante;

- **Exemplos:**

  char ch='D';

  int count=0;

  float pi=3.141;

  - **Não se garante a iniciação automática de variáveis declaradas sem iniciação.**

Lógica de Programação em C
Constantes

- São valores mantidos fixos pelo compilador
- Constantes dos tipos básicos

Tipo de Dado	Exemplos de Constantes
char	'b' '\n' '\0'
int	2 32000 -130
long int	100000 -467
short int	100 -30
unsigned int	50000 35678
float	0.0 23.7 -12.3e-10
double	12546354334.0 -0.0000034236556

Lógica de Programação em C
Constantes

- Constantes dos tipos básicos
  - Hexadecimais iniciam com 0x
  - Octais iniciam com 0

Constante	Tipo
0xEF	Constante Hexadecimal (8 bits)
0x12A4	Constante Hexadecimal (16 bits)
03212	Constante Octal (12 bits)
034215432	Constante Octal (24 bits)

- Constantes strings
  - ‘t’é diferente de “t”
- Constantes de barra invertida

Constantes Enumerações

- **Informa ao compilador quais valores uma variável pode assumir.**
- **Lista de valores inteiros constantes**

  enum boolean { NO, YES };

  - **NO = 0 e YES = 1**
- **Admite especificação dos valores inteiros**

```
enum escapes { BELL = '\a', BACKSPACE = '\b', TAB = '\t',
    NEWLINE = '\n', VTAB = '\v', RETURN = '\r' };
enum months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG,
    SEP, OCT, NOV, DEC };
        /* FEB = 2, MAR = 3, etc. */
```

Constantes Enumerações

```
#include <stdio.h>
enum dias_da_semana {segunda, terca, quarta, quinta,
   sexta, sabado, domingo};

Int main (void) {
   enum dias_da_semana d1,d2;
   d1=segunda;
   d2=sexta;
   if (d1==d2) {
       printf ("O dia e o mesmo.");
   }else {
       printf ("São dias diferentes.");
   }
   return 0;
}
```

Operadores Aritméticos e de Atribuição
Operadores Aritméticos

Operador	Ação
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto de divisão (de inteiros)
++	Incremento (inteiro e ponto flutuante)
--	Decremento (inteiro e ponto flutuante)

- **Divisão de inteiros é com truncamento.**
- **Operadores de incremento e de decremento podem ser pré-fixados ou pós-fixados**

Operadores Aritméticos e de Atribuição
Operadores Aritméticos – Exemplo

```
int a = 17, b = 3;
int x, y;
float z = 17. , z1, z2;
x = a / b;
y = a % b;
z1 = z / b;
z2 = a/b;
```

- **Quais os valores de x, y, z1 e z2 ?**

Operadores Aritméticos e de Atribuição
Operador de Atribuição

- **Operador de atribuição =**
  - **Atribuição com efeito colateral.**

    ```
    x=y=z=1.5;
    if (x==w) ...
    ```

  - **Há conversão de tipo.**

Lógica de Programação em C
Operadores Relacionais e Lógicos

Operador	Ação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Operador	Ação
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

Operadores Relacionais e Lógicos
Operadores Lógicos Bit a Bit

Operador	Ação
&	AND
|	OR
^	XOR (OR exclusivo)
~	NOT
>>	Deslocamento de bits à direita
<<	Deslocamento de bits à esquerda

- Forma geral do operador de deslocamento
  - *valor>>número_de_deslocamentos*
  - *valor<<número_de_deslocamentos*

Lógica de Programação em C
Expressões

- **São combinações de variáveis, constantes e operadores.**
  - **Conversão de tipos para o maior tipo:**
    1. Todos os **char** e **short int** são convertidos para **int**.
    2. Todos os **float** são convertidos para **double**.
    3. Para pares de operandos de tipos diferentes:
       - se um deles é **long double** o outro é convertido para **long double**;
       - se um deles é **double** o outro é convertido para **double**;
       - se um é **long** o outro é convertido para **long**;
       - se um é **unsigned** o outro é convertido para **unsigned**.

Expressões
Expressões de Atribuições

- **Expressões que podem ser abreviadas**
  - **op= onde op = {+,-,*,/,%,<<,>>,&,^,|}**
    - **exp1 op= exp2 equivale a exp1 = (exp1) op (exp2)**

      x += 10;
- **Expressão Ternária (ou Condicional)**
    - **exp1 ? exp2 : exp3.**

      z = (a>b) ? a:b;

Expressões de Atribuições
Expressões que Podem ser Abreviadas

Expressão Original	Expressão Equivalente
x=x+k;	x+=k;
x=x-k;	x-=k;
x=x*k;	x*=k;
x=x/k;	x/=k;
x=x>>k;	x>>=k;
x=x<<k;	x<<=k;
x=x&k;	x&=k;
etc...	

Expressões
Encadeamento de Expressões

- **Operador Vírgula**
  - **Expressões são executada seqüencialmente.**
    - **Retorna o valor da expressão mais a direita.**

      x=(y=2,y+3); **/* retorna 5 */**
  - **Em laço de for**

```
#include<stdio.h>
int main() {
int x, y;
for (x=0 , y=0 ; x+y < 100 ; x++ , y++)
   /* Duas variáveis de controle: x e y . Foi atribuído o valor zero a
   cada uma delas na inicialização do for e ambas são
   incrementadas na parte de incremento do for */
printf("\n%d ", x+y);
   /* o programa imprimirá os números pares de 2 a 98 */
}
```

Expressões
Tabela de Precedência do C

Operadores	Associatividade	Prioridade
() [] -> .	esquerda para a direita	1
! ~ ++ -- - (tipo) * & sizeof	direita para a esquerda	2
* / %	esquerda para a direita	3
+ -	esquerda para a direita	4
<< >>	esquerda para a direita	5
< <= > >=	esquerda para a direita	6
== !=	esquerda para a direita	7
&	esquerda para a direita	8
^	esquerda para a direita	9
|	esquerda para a direita	10
&&	esquerda para a direita	11
||	esquerda para a direita	12
?:	direita para a esquerda	13
= += -= /= *= %= >>= <<= &= ^= |=	direita para a esquerda	14
,	esquerda para a direita	15

Lógica de Programação em C
Modeladores (Cast)

- **Aplicado a expressão, força o resultado ser do tipo especificado.**
- **Forma geral**

  (tipo)expressão

  – **Existe Diferença?**

  f=(**float**) (num/7);

```
#include <stdio.h>
int main ()
{
int num;
float f;
num=10;
f=(float) num/7;
   /* Uso do modelador. Força a
    transformação de num em um
    float */
printf ("%f",f);
return (0);
}
```

Lógica de Programação em C
Controle de Fluxo

- Especifica a ordem em que as operações serão executadas na linguagem
  - Implícito (definido pelo tradutor)
    - Linguagem procedural
      - Esquerda para a direita
      - De cima para baixo
      - Operadores
        - Prioridade de avaliação e associatividade
  - Explícito (definido pelo programador)

Lógica de Programação em C
Declarações e Blocos

- As chaves { e } são usadas para agrupar declarações e comandos formando um *comando composto* ou *bloco.*
  - Sintaticamente equivalem a um comando simples.
  - Comando composto
    - Conjunto de comandos simples ou compostos.
  - Bloco
    - Conjunto de declarações e comandos
  - Não existe ponto-e-vírgula após o fecha chave.

Lógica de Programação em C
if-else

- **Expressa decisões.**
- **Sintaxe geral**

```
if (expressão)
    comando1      /* finaliza com ponto e vírgula */
else
    comando2
```

- **Comentários**
  - não existe o delimitador then.
  - O comando **else** é opcional.

if-else
Ausência do Comando Else

Regra implicita

- associa com o **if** sem **else** prévio mais próximo.

```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

Use { e } para definir a associação desejada

```
if (n > 0) {
    if (a > b)
        z = a;
}
else
    z = b;
```

If-else
Otimização

Otimizado

if (*expressão*)

if (!*expressão*)

condição ? expressão1:
    expressão2;

Não otimizado

if (*expressão* != 0)

if (*expressão* == 0)

if *(condição)*
    expressão1;
else
    expressão2;

Lógica de Programação em C
else-if

- **Expressa decisões múltiplas**
- **Sintaxe geral**

if (*expressão1*)
    comando1
else if (*expressão2*)
        comando2
else if (*expressão3*)
        comando3
else if (*expressão4*)
        comando4
else
    comando5

- **Comentários**
  - As expressões são avaliadas e se uma for verdadeira, o comando associado é executado e termina todo o processo.
  - o último **else** corresponde a uma situação *default* não disparada por nenhum teste anterior.
    - **Pode ser a indicação de uma condição impossível.**

else-if Exemplo

```
                                /* Pesquisa binaria: acha x in v[0] <= v[1] <= ... <= v[n-1] */
int bin (int x, int v[], int n)
{
    int esq, dir, meio;
    esq = 0;
    dir = n - 1;
    while (esq <= dir) {
        meio = (esq+dir)/2;
        if (x < v[meio])
        dir = meio - 1;
        else if (x > v[meio])
        esq = meio + 1;
        else                            /* achou referencia da chave */
        return meio;
    }
    return -1;                          /* nao achou chave */
}
```

Lógica de Programação em C
switch

- **Testa se expressão bate com um dos valores inteiros pré-estabelecidos**
- **Sintaxe geral**

  switch (*expressão*) {

  case *expr_const* : *comandos*

  case *expr_*const : *comandos*

  default: *comandos*

  }

- **Comentários**
  - **case** pode ter um ou mais inteiros ou expressão int.
  - não pode haver repetição.
  - **default** é executado se nenhum outro **case** o for.
  - **default** é opcional.
    - sem casamento, nenhuma ação é executada.
  - podem aparecer em qualquer ordem.

switch
Exemplo

```
#include<stdio.h>
int  nbranco = 0,                                   /* numero de espaco branco  */
         ndigito[10] = {0,0,0,0,0,0,0,0,0,0},       /* numero de digitos        */
         noutro = 0;                                /* caracteres de outros tipos   */
int main ()
{
    int    c,                                       /* caracter a ser lido por getchar */
           i;                                       /* variavel de trabalho (laco for) */
    while ( (c = getchar()) != EOF)
      switch (c) {
      case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7': case '8': case '9':
        ndigito[c-'0']++; break;
      case ' ': case '\n': case '\t':
        nbranco++; break;
      default:
        noutro++; break;
      }
    printf ("digitos \n "); for (i = 0; i < 10; ++i) printf ("[%d]=%d  ",i,ndigito[i]);
    printf ("\nespaco branco = %d, outro = %d\n",nbranco,noutro); return 0;
}
```

Lógica de Programação em C
while

- **Laço de repetição *executa apenas se verdadeiro*.**
- **Sintaxe geral**

  while (*expressão*)

  comando
- **Comentários**
  - **comando é opcional.**
  - **o laço pode ser infinito.**

    while (1) { .... }
  - **a partícula *do* não pode ser usada.**

Lógica de Programação em C
for

- **Laço de repetição *executa apenas se verdadeiro generalizado*.**
- **Sintaxe geral**

  for (*expr1*; *expr2*; *expr3*)

  comando
- **Comentários**
  - **Em geral, expr1 e expr3 são atribuições ou funções.**
  - **Em geral, expr2 é uma expressão relacional.**

for
Exemplos

- **Laço infinito (laço de espera ocupado)**

```
for (;;) {
...
}
```

- **Laço sem conteúdo (gera um *delay*)**

```
for (iniciação;condição;incremento);
```

- **Múltiplas expressões**
  - **Uso do operador vírgula.**

for
Exemplo de Uso do Operador Vírgula

```
#include <string.h>
                                /* reverse: inverte string s no lugar */
void reverse (char s[])
{
    int c, i, j;
    for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
            c = s[i];
            s[i] = s[j];
            s[j] = c;
    }
}
```

Lógica de Programação em C
do-while

- **Laço de repetição executa e testa após.**
- **Sintaxe geral**

  do

  comando

  while (*expressão*);
- **Comentários**
  - **Se expressão for falsa, a execução encerra.**
  - **Executa no mínimo uma vez.**

Lógica de Programação em C
break e continue

- **break**
  - Prover uma forma de se sair imediatamente do laço mais interno envolvendo o **break** ou de um comando **switch**.
    - Aplica-se a laço de **for**, **while** ou **do**.
  - Sua ação corresponde a um desvio incondicional
    - Tipicamente é usado com o comando **switch**.

```
/* trim: remove branco, tabs e LF
     do fim de strings */
int trim(char s[])
{
int n;
for (n = strlen(s)-1; n >= 0; n--)
    if (s[n] != ' ' && s[n] != '\t' &&
    s[n] != '\n') break;
s[n+1] = '\0';
return n;
}
```

Lógica de Programação em C
break e continue

- **continue**
  - Causa o início da próxima iteração de laço **for**, **while** ou **do** que o envolve
    - Não se aplica a **switch**.
    - No **while** ou **do**, a parte do teste é executada.
    - No **for,** o passo do incremento é executado.
  - A execução do laço continua normalmente, não sendo interrompido como no **break**.

```
for (i = 0; i < n; i++) {
    if (a[i] < 0)
        /* pula os negativos */
        continue;
    .....
}
```

Lógica de Programação em C
goto e Rótulos

- **Desvio incondicional**
- **Sintaxe geral**

nome_do_rótulo:
....
goto *nome_do_rótulo;*
....

goto *nome_do_rótulo;*
....
nome_do_rótulo:
....

- **Comentários**
  - **O rótulo e o goto devem estar na mesma função.**
  - **O rótulo é um identificador.**

goto e Rótulos
Situações Onde Usar

- Donald Knuth mostrou situações em que é preferível usar **goto**, mesmo em programas estruturados.
  - Saída de laços aninhados:
    - Razão de simplicidade e legibilidade do código:
      - o **break** permite sair apenas do laço mais interno.

```
for ( ... ) {

    for ( ... ) {
        ...
        if (disastre) goto error;
    }
...
}
error:
```

goto e Rótulos
Situações Onde Usar

- Código mais limpo

```
for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
      if (a[i] == b[j])
        goto achou;
/* não achou elemento
   comum aos vetores */
...
achou:
      /* achou a[i] == b[j] */
...
```

- Código equivalente sem goto

```
achou = 0;
for (i = 0; i < n && !achou; i++)
    for (j = 0; j < m && !achou; j++)
        if (a[i] == b[j]) achou = 1;
if (achou)
    /* achou a[i-1] == b[j-1] */
...
else
    /* não achou elemento comum
    aos vetores */
...
```

Palavras Reservadas

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

Introduzidas pelo padrão C99:

inline
_Bool **// requer #include stdbool.h**
_Complex **// requer #include complex.h**
_Imaginary