Linguagem C

ESTRUTURA BÁSICA DE UM PROGRAMA C

✓ consiste em uma coleção de funções

✓ forma geral

Main ( ) → primeira função a ser executada
{ → início da função
} → fim da função

✓ obs: para o compilador

» Main ( ) { }

Main ( )
{ }

» Main
(
)
{
}

São programas idênticos

A FUNÇÃO MAIN ( )

✓ tem que existir em algum lugar
✓ marca o início da execução

ex:

```
Main( )
{
   printf  (“meu primeiro programa”);
}
```

obs:

✓ toda instrução deve ser encerrada por ;
✓ printf é uma função, note um '( )' após o nome

novamente:

Main ( ) { printf ("meu primeiro programa"); }

A FUNÇÃO PRINTF ( )

✓ função de E / S

Note que:

✓ uma função pode receber uma informação (argumento)
✓ printf("meu primeiro programa") ;

↓

meu primeiro programa

SINTAXE:

printf ("expressão de controle", lista de argumentos);

ex:

```
Main ( )
{
  printf ("o número %d", 2);
}
```

↓

código de formatação

MAIS PRINTF

```
Main ( )
{
  Printf ("%s está a %d milhões de milhas \ n do sol", "vênus",
     67);
}
```

saída: vênus está a 67 milhões de milhas
do sol

✓ obs: \ n é um código especial que indica uma mudança de linha

```
Main ( )
{
   printf  (" a letra %c", 'a');
   printf  (" vem antes de %c", 'b');
}
```

saída: a letra a vem antes de b

CARACTERES DE CONTROLE

\n	nova linha
\r	"enter"
\t	tabulação (tab)
\b	retrocesso
\"	aspas
\ \	barra

CÓDIGO DE FORMATAÇÃO

%c	caracter
%d	decimal
%e	notação científica
%f	ponto flutuante
%o	octal
%s	cadeia de caracteres (string)
%x	hexadecimal

CONSTANTES E VARIÁVEIS

Constante: "objeto" que tem valor fixo e inalterável

ex: 'c', 8, "primeiro programa"

Uso:

```
Main ( )
{
   printf  (" o número %d", 2);
}
```

```
Main ( )
{
   printf  (" o número 2");
}
```

Variáveis:

✓ um "objeto" que pode assumir diferentes valores

- ✓ espaço de memória de um certo tipo de dado associado a um nome para referenciar seu conteúdo

ex:

```
Main ( )
{
   int idade;
   idade = 30;
   printf (“ A idade mínima é : %d”, idade);
}
```

Instrução para reservar uma quantidade de memória para um certo tipo de dado, indicando o nome pelo qual a área será referenciada

Na sua forma mais simples:

- ✓ tipo nome-da-variável; ou
- ✓ tipo nome1, nome2, ... nomen;

ex:
```
int  a;
int  b;          ou        int   a, b;
```

ex 2:
```
char letra;
int número, idade;
```

ex3:
```
main ( )
{
   int      x;
   float   y;
   x = 3;
```

```
    y = 3 * 4.5;
    printf (“ %d * 4.5 = %f”, x, y);
}
```

TIPOS BÁSICOS

- ✓ determina um conjunto de valores e as possíveis operações realizadas sobre os mesmos
- ✓ informa a quantidade de memória (bytes)

tipo	bytes	escala
char	1	-128 a 127
int	2	-32.768 a 32.767
float	4	3.4e-38 a 3.4e+38
double	8	1.7e-308 a 1.7e+308
void	0	sem valor

Modificadores de tipos

Long ou Long int	(4 bytes)
Unsigned Char	(0 a 255)
Unsigned int	(0 a 65.535)

ex: inteiros

```
Main ( )
```

```
{
   unsigned    int     j = 65.000;
   int  i = j;
   printf (“ %d   %u”, i, j);
}
```

saída: -536 65.000

PORQUÊ ?

- ✓ na forma binária o bit 15 é 0 se o número for positivo e 1 se negativo

INICIALIZANDO VARIÁVEIS

- ✓ a combinação de uma declaração de variáveis com o operador de atribuição

```
Main ( )
{
  int     evento   = 5;
  char   corrida  = ‘c’;
  float   tempo    = 27.25;

  printf (“ o melhor tempo da eliminatória % c”, corrida);
  printf (“ \ n do evento %d  foi % f”, evento, tempo);
}
```

NOMES DE VARIÁVEIS

✓ quantos caracteres quiser (32)

✓ comece com letras ou sublinhado:
✓ seguidos de letras, números ou sublinhados

obs:

✓ 'C' é sensível ao caso:
peso <> Peso <> pEso

✓ não podemos definir um identificador com o mesmo nome que uma palavra chave

Palavras Chave:

auto	static	extern	int	long	if
if	do	default	while	do	etc

EXPLORANDO A FUNÇÃO PRINTF

Tamanho de campos:

✓ é possível estabelecer o tamanho mínimo para a impressão de um campo

```
Main ( )
{
   printf ("os alunos são %3d \ n", 350);
   printf ("os alunos são %4d \ n", 350);
   printf ("os alunos são %5d \ n", 350);
```

```
}
```

Saída: os alunos são 350
os alunos são 350
os alunos são 350

Note:

```
Main ( )
{
    printf (“ %4.1f    \ n”, 3456.78);
    printf (“ %10.3f  \ n”, 3456.78);
}
```

Saída: 3456.8
3456.780

A FUNÇÃO SCANF( )

✓ função de E / S
✓ complemento de printf( )

Sintaxe:

scanf(“expressão de controle”, lista de argumentos)

expressão: %* lista: &variável

ex: Main ( )
{
int num;
scanf(“ %d”, &num);
}

```
Main ( )
{
  char letra;
  scanf (“%c”, &letra);
}
```

O OPERADOR DE ENDEREÇO (&)

1 Kbyte = 1024 endereços

1 Mbyte = 1024 x 1024 = 1.048.576 endereços

8 Mbyte = 8 x 1024 x 1024 = 8 x 1.048.576
= 8.388.608 endereços
↓
0 até 8.388.607

1 Gbyte = 1024 x 1024 x 1024 = 1.073.741.824

- ✓ um endereço de memória é o nome que o computador usa para identificar uma variável
- ✓ toda variável ocupa uma área de memória e seu endereço é o do primeiro byte por ela ocupado

Ex :

inteiro → 2 bytes
float → 4 bytes
char → 1 byte

Quando usamos & precedendo uma variável, estamos falando do endereço da mesma na memória

Ex:

```
Main ( )
{
   int num;
   num = 2;
   printf ("valor = %d, endereço = %p", num,
&num);
}
```

Saída: valor = 2, endereço = 1230
↑

varia conforme máquina / memória

CÓDIGO DE FORMATAÇÃO SCANF( )

%c → caracter
%d → inteiro
%e → número ou notação científica
%f → ponto flutuante
%o → octal
%x → hexadecimal
%s → string (cadeia de caracteres)
%lf → double

Ex:

```
Main ( )
{
    char a ;

  printf ( “digite um caracter” );
  scanf ( “ % c”, &a );
  printf (“ \n %c = %d em decimal”,  a, a);
  printf (“%o  em octal, %x em hexadecimal”,  a,  a);
}
```

Digitando **m**:

m = 109 em decimal, 155 em octal, 6d em hexadecimal

FUNÇÕES GETCHE( ) E GETCH( )

- ✓ A função scanf obriga que a tecla <enter> seja pressionada após a entrada dos dados
- ✓ a biblioteca de C oferece funções que lêem dados sem esperar <enter>

getche( ): lê um caracter do teclado ecoando-o na tela

getch( ): lê um caracter do teclado sem ecoá-lo na tela

Ex :

```
main ( )
{
   char ch;
   printf ( “digite um caracter”);
   ch = getche( );
   printf ( “\n todos sabem que você digitou %c”,
ch);
}
```

Executando:

digite um caracter: a
todos sabem que você digitou a

USANDO GETCH ...

```
Main ( )
{
   char ch;
    ch = getch( );
    printf (“ \ n somente agora saberemos”);
     printf (“que você digitou %c”, ch);
}
```

Executando:

Digite um caracter:
Somente agora saberemos que você digitou
b

Operadores

Aritméticos

✓ binários: **=** **+** **-** ***** **/** **%**

✓ unário: **-**

Ex:

```
Int a, b;

b = 3;                          a = b + 2;
b = a * b;                      b = 7 % 2;
```

Nota:

a = 2000; ← **válido**

2000 = a; ← **inválido**

↓

constante

Mais um Exemplo ...

```
Main ( )
{
   int   nota, conceito;

   printf ( “entre com a nota e o conceito”);
   scanf ( “ %d  %d”, &nota, &conceito);
   printf ( “sua nota final é %d”,  nota * conceito);
}
```

Mais um exemplo ...

```
Main ( )
```

```
{
   int   resto, divisor, dividendo;

   printf("entre com 2 números");
   scanf(" %d   %d ,  &dividendo,  &divisor);
   resto  =   dividendo % divisor;
   printf("o resto da divisão inteira de %d", dividendo);
   printf("por  %d  =  %d", divisor, resto);
}
```

Saída:

```
   entre com 2 números  10  4
   o resto da divisão inteira de  10  por  4  =  2
```

OPERADORES DE INCREMENTO (++) E DECREMENTO (--)

✓ Incrementam / decrementam uma unidade de seu operando

✓ modos distintos
- pré - fixado
- pós - fixado

ex: `int  n;`
`n = 0;`
`n++;` $\Rightarrow$ `n  =  n + 1;` $\rightarrow$ `n = 1`

Se o operador é usado em uma instrução:

`n = 5;`
`x = n++;` $\rightarrow$ `x = 5` (usa a variável e depois incrementa)
`n = 6`

ex: n = 5;
x = n++ * 3; → x = 15 n = 6

ex: n = 5;
x = ++n * 3; → n = 6 x = 6 * 3 = 18

ex: n = 6;
x = n-- / 2; → x = 6 / 2 = 3 n = 5

MAIS EXEMPLOS:

Ex: n = 5;
x = --n / 2; → n = 4 x = 4/2 = 2

```
Main( )
{
     int num = 0;
     printf (“ %d”, num);
     printf (“ %d”, num++);
     printf (“ %d”, num);
}
```

Saída: 0 0 1

```
Main ( )
{
    int num = 0;
    printf (“ %d”, num);
    printf (“ %d”, ++num);
    printf (“ %d”, num);
}
```

Saída: 0 1 1

E se no lugar de num++ e ++num tivéssemos num-- e --num, qual seria a saída?

PRECEDÊNCIA

- (unário)
*** / %**
+ - (binário)
=

Ou seja: x = 3 * a - b $\Rightarrow$ (3 * a) - b
x = y = 5 % 2 $\rightarrow$ x = (y = (5 % 2))

Agora: **- ++ --**
*** / %**
+ - (binário)
=

Ou seja: x = 3 * a++ - b $\Rightarrow$ (3 * (a++)) - b
y = 3 * --a - b $\rightarrow$ (3 * (--a)) - b
z = a * b++ $\rightarrow$ a * (b ++)

Obs: ++, -- só podem ser usados com variáveis

Erro: (a * b) ++;
5++;

CUIDADO COM PRINTF( )

Ex: n = 5;
printf (“ %d %d %d \n”, n, n + 1, n++);

saída: 5 6 5 (avaliação feita à esquerda)
saída: 6 7 5 (avaliação feita à direita)

OPERADORES ARITMÉTICOS DE ATRIBUIÇÃO

- ✓ **+=, -=, *=, /=, %=**
- ✓ atribuir um novo valor à variável dependendo do operador e da expressão a direita

A sintaxe:
x op= exp é equivalente a **x = (x) op (exp)**

Ex:

i += 2 → i = i + 2;

x *= y + 1 → x = x * (y + 1)

t /= 4 → t = t / 4

p %= 6 → p = p % 6

h -= 3 → h = h - 3;

↓

produz código de máquina mais eficiente

OPERADORES RELACIONAIS

✓ Usados para comparações

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

✓ Em C não existe o tipo "booleano"

0 → falso

Valor diferente de 0 é verdadeiro

Ex:

```
main ( )
{
   int verdadeiro, falso;
   verdadeiro = (15 < 20);
   falso  =  (15 == 20);
```

```
   printf ("Verd. = %d, falso = %d", verdadeiro,
falso);
}
```

saída: Verd. = 1 falso = 0

PRECEDÊNCIA

Agora: - ++ --
*** / %**
+ - (binário)
< > <= >=
== !=
= += -= *= /= %=

Comentários

- ✓ São informações acrescentadas ao código para facilitar sua compreensão
- ✓ São ignoradas pelo compilador (não faz parte do código objeto)
- ✓ Começa com **/*** terminando com ***/**

Ex:
```
/*  isto é um exemplo de comentário  */
main ( )
```

```
{
   printf (“ apenas um exemplo”);
}
```

COMENTÁRIOS

Obs: /* ou */ dentro de um comentário é Inválido:

Ex: /* isto não é um /* comentário */

É válido: / * comentário em
mais de uma linha */

ESTRUTURAS DE REPETIÇÃO

```
Main ( )
{
  printf (“1”);
  printf (“2”);
    : : :
  printf (“10”);
}
```

saída: 1 2 3 4 ... 10

Como imprimir os 1000 1os números a partir de 1?

Solução 1:

```
Main ( )
{
    printf (“1”);
    printf (“2”);
      :  :  :
    printf (1000);
}
```

✓ for, while, do-while
✓ repetir uma sequência de comandos

```
Main ( )
{
  int num;
  for (num = 1;  num <= 1000;  num++)
      printf (“ % d”, num);
}
```

Saída: 1 2 3 ... 1000

Forma geral:

for (inicialização; teste; incremento)

instrução; } corpo do laço

Na forma mais simples:

✓ Inicialização:
expressão de atribuição
sempre executada uma única vez

✓ Teste:
condição que controla a execução do laço
é sempre avaliada a cada execução
verdadeiro → continua a execução
falso → para a execução

✓ Incremento:

define como a variável de controle será alterada

é sempre executada após execução do corpo do laço

IMPRIMINDO NÚMEROS PARES

```
Main ( )
{
    int número;
    for  ( número = 2;  número < 10;  número += 2
  )
      printf (“ %d”, número);
```

```
}
```

Saída 2 4 6 8

FLEXIBILIDADE

Qualquer expressão de um laço for pode conter várias instruções separadas por vírgula

Ex:

```
/*  imprime os números de  0  a 100 em
                incremento de  2   */
Main ( )
{
      int x, y;

      for (x = 0, y = 0;   x + y   <=  100;  x++, y++)
          printf ("%d",  x + y);
}
```

Ex:

```
/* imprime as letras do alfabeto  */
Main ( )
{
      char ch;
      int i;

      for (i = 1, ch = 'a';    ch  <=   'z';    ch++,  i++)
```

```
        printf ("%d.a  letra  =  %c  \n",  i,  ch);
}
```

Note o uso de funções nas expressões do laço ...

```
Main ( )
{
  char ch;
  for ( ch = getch ( );    ch  !=  'x';   ch  =  getch( )
)
     printf ( "%c", ch + 1);
}
```

Obs: Qualquer uma das 3 expressões pode ser omitida, permanecendo apenas os ";"

Reescrevendo o exemplo anterior:

```
Main ( )
{
      char ch;
      for ( ;  (ch = getch ( ))  !=  'x';  )
        printf ( " %c" , ch + 1);
}
```

CUIDADO

Se a expressão de teste não estiver presente é

considerada sempre verdadeira

```
Main ( )
{
   for ( ; ; )   printf ( "\n estou  em  loop  infinito");
}
```

```
Main ( )
{
      int i;
      for ( i = 0;   i  <  100;   i++ )  ;   printf ( i );
}
```

corpo vazio

Saída: 100

MÚLTIPLAS INSTRUÇÕES

Quando duas ou mais instruções estiverem fezendo parte do corpo do laço, estas devem ser colocadas entre { } (bloco de instruções)

```
Main ( )
{
    int num, soma;
```

```
    for (num =  1,  soma = 0;   num  <=  3;  num++)
    {
      soma  +=  num;
       printf ("nº  =  %d   soma = %d \n", num,
    soma);
    }
}
```

Saída: nº = 1 soma = 1
nº = 2 soma = 3
nº = 3 soma = 6

LAÇOS ANINHADOS

✓ Quando um laço está dentro de outro, dizemos que o laço interior está aninhado

```
Main ( )
{
      int i, J;
      for (i = 1;   i <= 3;   i++)
         for (J = 1;  J <= 2;  J++)
            printf ("%d → %d \n",  i,  J);
}
```

Saída: $1 \rightarrow 1$ $2 \rightarrow 1$ $3 \rightarrow 1$
$1 \rightarrow 2$ $2 \rightarrow 2$ $3 \rightarrow 2$

Qual a saída do seguinte programa:

```
Main ( )
{
  int  i, J;
  for (i = 1;  i <= 3;  i++);
    for (J = 1;  J <= 2;  J++)
         printf (" % d → % d", i, J);
}
```

↓

É visto pelo compilador como:

```
Main ( )
{
  int  i, J;
  for (i = 1; i  <= 3;  i++);
  for (J = 1;  J  <=  2;  J++)
     printf ("%d  →  %d", i, J);
}
```

Saída: $4 \rightarrow 1$
$4 \rightarrow 2$

\

O LAÇO WHILE

Forma geral:

while (**expressão de teste**)
Instrução;

- ✓ “Instrução” só é executada se “expressão de teste” for verdadeira (!= 0)
- ✓ Expressão é sempre avaliada até que se torne falsa (= 0)

Ex:

```
 main ( )
 {
int num;

num = 0;
while (num < 3)
     printf (“ %d”,  num++);
 }
```

Saída: 0 1 2

Obs: O corpo de um **While** pode ter:

uma única instrução,

várias instruções entre chaves ou

nenhuma instrução

Ex:

```
main ( )
{
  int  num = 1, soma = 0;
  printf (“ a soma de: ”);
  while (num <= 3)   {
 soma  +=  num;
 printf (“ %d”, num);
  }
  printf (“ = % d”, soma);
}
```

Saída: A soma de: 1   2   3 = 6

Ex:

```
main ( )
{
     long  tm;
     int  contador = 0;
     printf (“%d”,  tm = time (0));
    while (contador++   <=   100.000);
    printf (“esperou o tempo: %ld”, time (0) -
tm);
}
```

WHILE X FOR

FOR:

✓ sabe-se a princípio o número de interações,

✓ o número de repetições é fixo;

WHILE:

✓ não se sabe a princípio o número de interações,

✓ o laço pode terminar inesperadamente;

Ex: Contar o número de caracteres de uma frase até que <enter> seja digitado

```
main ( )
{
  int cont = 0;
  printf (“ digite uma frase: \n”);
  while (getche( ) !=  ‘\ r’)
  cont++;
  printf (“\n o número de caracteres é %d”,
cont);
}
```

Note que:

```
for (inicializa;  teste;  incremento)  =  inicializa;
     instrução;                                  while
    (teste)  {

    instrução;

    incremento;
                                                  }
```

✓ While's dentro de um laço While

```
main ( )
{
    int  num,   vezes  =  1;
    char  continua = 's';
    while (continua  ==  's')   {
      printf (" \n digite um nº entre 1 e 1000");
      scanf (" %d",  &num);
      while (num  !=  50) {
     printf ("%d  incorreto.", num);
     printf(" Tente novamente \n");
     scanf ("%d",  &num);
     vezes++;
      }
      printf ("\n acertou em %d tentativa(s)",
    vezes);
       printf ("\n joga novamente? (s / n):");
      continua  =  getche( );
    }
}
```

O LAÇO DO-WHILE

- ✓ Cria um ciclo repetitivo até que a expressão seja falsa (zero)

- ✓ Similar ao laço While
  a diferença está no momento em que a condição é avaliada

Forma Geral:

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

Observação:

- ✓ As chaves são opcionais se apenas um comando está presente

Ex:

```
/ * testa a capacidade de adivinhar uma letra * /
main ( )
{
  char  ch;
   int   tentativas;
  do  {
```

```
    printf ("digite uma letra");
    tentativas = 1;
    while  ( (ch = getch ( ) )   !=  't')   {
          printf ("%c  é  incorreto \n", c);
          tentativas++;
            printf ("tente novamente \n");
   }
   printf ("%c é correto", c);
   printf ("acertou em %d vezes", tentativas);
   printf ("continua? (s / n):");
    }  while  (getche( )   ==   's');
  }
```

- ✓ Estimativa de 5%
- ✓ Evita duplicação de código
- ✓ Executar o laço (pelo menos uma vez) mesmo que a condição seja falsa

O COMANDO BREAK

- ✓ O comando Break pode ser usado em qualquer estrutura de laço em C:

causa a saída imediata do laço

- ✓ Quando estiver presente em laços aninhados afetará somente o laço que o contém (e os internos, obviamente)

ex:

```
main ( )
{
   int num;

   while (1)  {
  printf ( "\n digite um número");
  scanf ("%d", &num);
  printf (" 2  *  %d  =  %d", num, 2 * num);
  break;
   }
}
```

O COMANDO CONTINUE

- ✓ O comando Continue força a próxima interação do laço (ignorando o código que estiver abaixo)

✓ No caso de While, Do-While, o comando Continue faz com que o controle vá direto para o teste condicional

✓ No caso de um Laço For:

primeiro o incremento é executado

depois o teste condicional

Obs:

Deve-se evitar o comando Continue, pois dificulta a manutenção de um programa

ESTRUTURAS DE DECISÃO

✓ Permitir testes para decidir ações alternativas
✓ IF, IF - ELSE, SWITCH e Operador Condicional (?:)

O COMANDO IF

Forma Geral: If (condição)
instrução;

```
Main ( )
{
   char ch;
   ch = getche ( );
   If  (ch == 'p')
```

```
      printf (“você pressionou a tecla p”);
}
```

MÚLTIPLAS INSTRUÇÕES

Forma:

```
    If  (condição)  {
        comando 1;
        comando 2;
    }
```

```
Main ( )
{
    If  (getche()) == ‘p’ ) {
      printf (“ você digitou p”);
      printf (“ pressione outra tecla ”);
      getche( );
    }
}
```

IF ANINHADOS

✓ Se um comando If está dentro de outro If, dizemos que o If interno está aninhado

```
Main ( )
{
```

```
    char ch;
    printf (“ digite uma letra entre A e Z”);
    ch = getche ( );
    If (ch >= ‘A’)
      If (ch < = ‘Z’)
        printf (“ você acertou”)
}

Main ( )
{
    char ch;
    printf (“ digite uma letra entre A e Z”);
    ch = getche ( );
    If   ((ch >= ‘A’)  &&  (ch < = ‘Z’))
       printf (“ você acertou”)
}
```

O COMANDO IF - ELSE

- ✓ O comando If só executa a instrução caso a condição de teste seja verdadeira, nada fazendo se a expressão for falsa
- ✓ O comando else executará um conjunto de instruções se a expressão de teste for falsa

```
Forma Geral:If (condição)
                  instrução
            else
                  instrução

Main ( )
{
    If (getche ( ) == ‘p’)
```

```
        printf (“ você digitou p”);
    else
        printf (“ você não digitou p”);
}
```

IF - ELSE ANINHADOS

Forma Geral:If (condição 1)
instrução
else If (condição 2)
instrução
else if (condicao 3) ...

```
Main ( )
{
    int número;
    scanf (“ % d”, &número);
    If (número < 0)
      printf (“número menor que zero”);
    else If (número < 10)
      printf (“ número ≥ 0 e < 10”);
    else If (número < 100)
      printf (“número ≥ 10 e < 100)
    else
      printf (“número ≥ 100”);
}
```

Como o computador decide de qual If o else pertence?

```
Ex:  If (n > 0)
        If (a > b)
```

```
        z = a;
    Else
        z = b;
```

Quando z = b será executado ?

✓ else é sempre associado ao If mais interno (mais próximo)

Note a diferença:

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

OPERADORES LÓGICOS

✓ && , II, !

Ex:

```
(1  II 2)
(x  &&  y)
(a > 10)  &&   (x < 10)
! (x > 0)
(10 <= a)  && (a <= 100)
```

Cuidado: (10 <= a <= 100) == ((10 <= a) <= 100)

EXEMPLOS:

✓ If (10 < a) && (a < 100) / * 10 < a < 100 * /

✓ If (10 < a) II (a == -1)

Contando caracteres e dígitos de uma frase

```
main ( )
{
    char c;
    int car = 0,  dig = 0;

    printf (“ digite uma frase encerre com <enter>”);
    while ( ( c = getche ( ) )  != ‘\r’ )  {
       car++;
        If ( ( c >= ‘0’)  &&   ( c < = ‘9’))
      dig++;
    }
    printf (“ número de caracteres %d”, car);
    printf (“ número de dígitos %d”, dig);
}
```

□□

Obs: lembre-se que 0 em C é falso e qualquer valor diferente de 0 é verdadeiro, logo:

✓ If (nota == 0) → If (!nota)

Precedência:

```
! - ++ - -
* / %              Aritméticos
+ -
< > <= >=          Relacionais
==  !=
&&
 II                Lógico
= += -= *= /= %=   Atribuição
```

O COMANDO SWITCH

✓ Forma de substituir o comando If - else ao se executar vários testes
✓ Similar ao If - else com maior flexibilidade e formato limpo

FORMA GERAL:

```
switch (expressão) {
    case constante 1:
      instruções; /* opcional */
      break;  /* opcional */
    case constante 2:
      instruções
```

```
      break;
    default:
      instruções
}
```

Expressão: tem que ser um valor inteiro ou caracter

Ex: uma calculadora

```
Main ( )
{
    char op;
    float num 1, num 2;

    while (1) {
       printf (" digite um n.o, um operador e um n.o");
       scanf (" %f  %c %f",   &num1,  &op,  &num2);
       switch (op) {
     case '+':
          printf (" = %f", num 1 + num 2);
          break:
     case  '-':
          printf (" = %f", num 1 - num 2);
          break;
     default:
          printf (" operador inválido");
        }
    }
}
```

O OPERADOR CONDICIONAL TERNÁRIO ?:

- ✓ Forma compacta de expressar uma instrução If - else

Forma Geral:

(Condição) ? expressão 1 : expressão 2

```
Max = (num1 > num2) ? num1 : num2;
```

Note:

```
If (num1 > num2)
    Max = num 1;
Else
    Max = num 2;
```

Exemplo:

```
ABS = (num < 0) ? - num : num;
```

FUNÇÕES / PROCEDIMENTOS

- ✓ Funções : abstrações de expressões
- ✓ Procedimentos: abstrações de comandos
- ✓ Dividir uma tarefa complexa em tarefas menores, permitindo esconder detalhes de implementação

✓ Evita-se a repetição de um mesmo código

Forma Geral:

```
Tipo   Nome (lista de parâmetros)
{
    corpo
}
```

PROCEDIMENTO

✓ “Funções” que não retornam valores

Ex:

```
void   desenha( )
{
   int i;
   for (i = 0;  i < = 10;  i++)
  printf (“-”);
}

Main ( )
{
   desenha ( );
   printf (“ usando funções”);
   desenha ( );
}
```

FUNÇÕES

Ex:

```
int   fatorial (int  n)
{
   int  i, resultado = 1;
   for ( i = 1;  i <= n;  i ++)
  resultado *=  i;
   return  resultado;
}

Main ( )
{
      printf (“ o fatorial de 4 = %d”, fatorial(4) );
      printf (“ o fatorial de 3 = %d”, fatorial(3) );
}
```

VARIÁVEIS LOCAIS

- ✓ Variáveis declaradas dentro de uma função são denominadas locais e somente podem ser usadas dentro do próprio bloco
- ✓ São criadas apenas na entrada do bloco e destruídas na saída (automáticas)

Ex:

```
void desenha ( )
{
   int  i, j;
     .  .  .
     .  .  .
}

main ( )
```

```
{
  int a;
  desenha();
  a = i; ← erro
  . . .
}
```

Ex 2:

```
void desenha ( )
{
   int i, j;
   . . .
   . . .
}
void calcula ( )
{
   int i, j;
   . . .
   . . .
}
```

i, j em desenha são variáveis diferentes de i, j em calcula.

VARIÁVEL GLOBAL

✓ Variável que é declarada externamente podendo ser acessada por qualquer função

Ex:

```
int i;

void desenha ( )
{
   int j;
   i = 0;
   . . .
}

void calcula ( )
{
   int m;
   i = 5;
   . . .
}
```

Exemplo

```
char  minúsculo( )
{
   char   ch = getche( );
   If ( (ch  >=  'A') &&  (ch  <=  'Z'))
         ch  +=  'a'  -  'A';
   return (ch);
}
```

O COMANDO RETURN

- ✓ Causa a atribuição da expressão a função,
- ✓ Forçando o retorno imediato ao ponto de chamada da função

Exemplo

```
char  minúsculo ( )
{
   char ch;
   ch = getche( );
   If  ( (ch >= 'A')  &&   (ch <= 'Z'))
      return (ch + 'a' - 'A');
   else
      return (ch);
}


Main ( )
{
      char letra;
      printf (" digite uma letra em minúsculo");
      letra = minúsculo ( );
      If   (letra  == 'a')        //   if  (minusculo( ) == 'a')
         printf ("ok");
}
```

- ✓ Note pelo exemplo anterior que a função minúsculo lê um valor internamente convertendo-o para minúsculo.

Como usar esta função se já temos uma letra e desejamos convertê-la para minúsculo?

PASSANDO DADOS PARA FUNÇÕES

- ✓ Passagem de parâmetro por valor - uma cópia do argumento é passada para a função
- ✓ O parâmetro se comporta como uma variável local

Ex:

```
void minúsculo (char ch)
          ↑ parâmetro formal
{
  If (( ch >= 'A')  (ch <= 'Z'))
     return (ch + 'a'-, 'A');
  else
     return (ch);
}
```

```
Main ( )
{
   printf (“ %c”, minúsculo (‘A’) );
                                    ↑ parâmetro real
}
```

Ex 2: Valor Absoluto

```
int  abs (int x)
{
   return ( ( x < 0 ) ? -x : x );
}

Main ( )
{
   int  num, b;
   printf (“ entre com um número  > o”);
   scanf (“ %d”,  &num );
   b = abs (num);
     .  .  .
     .  .  .
   printf (“ Valor absoluto de num = %d”, abs(num)
);
     .  .  .
   b  =  abs(-3);
}
```

PASSANDO VÁRIOS ARGUMENTOS

- ✓ Frequentemente uma função necessita de mais de uma informação para produzir um resultado
- ✓ Podemos passar para a função mais de um argumento

Ex 1:

```
float área_retângulo  (float   largura, float  altura)
{
    return (largura * altura);
}
```

Ex 2:

```
float potência (float  base,  int  expoente)
{
   int  i;    float  resultado = 1;
   If (expoente == 0)
     return 1;
   For (i = 1; i <= expoente; i++)
      resultado *= base
   return  resultado;
}
```

USANDO VÁRIAS FUNÇÕES

Calcular a seguinte sequência:

$S(x, n) = x/1! + x^2/2! + x^3/3! + ... + x^n/ n!$

Solução:

```
int  fat (int n)
```

```
    {
       int i, resultado = 1;
       for ( i = 1;  i <= n;  i ++)
      resultado *= i;
       return resultado;
    }
    float   potencia (float  base,  int expoente)
    {
       int i;          float  resultado = 1;
       If (expoente == 0)
         return 1;
       for (i = 1;  i <= expoente;  i++)
          resultado *= base;
          return resultado;
    }
```

```
float  serie (float x,   int n)
{
        int i;      float  resultado = 0;
        for ( i = 1;  i <= n; i++)
           resultado  +=  potência( x, i )  /  fat( i );
        return resultado;
}

void  main( )
{
```

```
    float x;
    int  termos;

    printf("entre com o numero de termos: ");
    scanf("%d", &termos);
    printf("entre com o valor de X: ");
    scanf("%f", &x);
    printf("O valor de série = %f ", serie(x, termos));
}
```

Arranjos

- ✓ tipo de dado usado para representar uma coleção de variáveis de um mesmo tipo
- ✓ estrutura homogênea

Ex: Ler a nota de 3 alunos e calcular a média

```
    int  nota0, nota1, nota2;

    printf("entre com a 1a. nota");
    scanf("%d", &nota0);
      :      :      :
    printf("média = %f", (nota0 + nota1 + nota2) / 3));
```

Problema: Calcular a média de 300 alunos.

Solução: Arranjo

- ✓ Arranjos:

Unidimensional (VETOR)
N-dimensional (MATRIZ)

- ✓ Informalmente:
  "arranjo é uma série de variáveis **do mesmo tipo** referenciadas por um único nome"
  cada variável é diferenciada por um índice

Ex:

```
int   nota [ 4 ];
```

↓

Vetor de inteiros

nota [0], nota [1], nota [2], nota[3]

Obs: tamanho m → índice 0 a (m - 1)

Exemplo

Contar o número de vezes que um dado caractere aparece em um texto

```
#define TAM  256

main( )
{
    int     i,  letras [ TAM ];
    char   simbolo;

    for ( i = 0;  i < TAM;  i++)
        letras [ i ] = 0;
```

```
    //  ler a sequencia ate  <enter>  ser
    pressionado while ( ( simbolo = getche( ) )  !=
    '\r' )
        letras [ simbolo ]++;

    for ( i = 0;  i < TAM;  i++)      {
        printf ( "o caracter %c  ",  i);
        printf ( "apareceu %d vezes",  letras [ i ] );
    }
}
```

Inicializando Arranjos

✓ Considere uma variável inteira **numero**

✓ Podemos inicializar a variável **numero**:

```
int   numero = 0;
numero = 0;
scanf ("%d", &numero);
```

✓ Dado um arranjo podemos inicializá-lo:

```
int   notas [ 5 ] = { 0, 0, 0, 0, 0 }
notas [0] = 0;  notas [1] = 0  ...  notas [4] = 0;
for ( i = 0 ; i < 5;  i++)   scanf ("%d", &notas [ i ] );
```

Obs: Dado int notas [10] podemos fazer:

```
notas [9] = 5;
notas [4] = 50;
```

as demais posições do vetor contêm "lixo"

Exemplo

Imprimir a média da turma e a nota de cada aluno.

```
#define N_ALUNOS  40

main( )
{
    int    i;
    float notas [ N_ALUNOS ],  media = 0;

    for ( i = 0; i < N_ALUNOS; i++ ) {
      printf ("entre com a nota %d", i+1);
      scanf ("%f", &notas[ i ]);
      media += notas [ i ];
    }

    printf (" Média = %f  \n", media / N_ALUNOS);

    for ( i = 0; i < N_ALUNOS; i++ ) {
      printf ("\n Nota do aluno %d = ", i+1);
      printf ("%f  \n", notas[ i ]);
    }
}
```

Trabalhando com um número desconhecido de elementos

- ✓ em 'C' não existe declaração de arranjo dinâmico
- ✓ o tamanho de um arranjo tem que ser determinado em tempo de compilação

```
Ex:     int  alunos;
        int  notas [ alunos ];
             :     :     :
        printf ("entre com o número de alunos");
        scanf ("%d", &alunos);
```

NÂO É ACEITO !!!

✓ Solução: declarar um arranjo que suporte um número máximo de elementos

```
Ex:     int  alunos;
        int  notas [ 70 ];
             :     :     :
        printf ("entre com o número de alunos");
        scanf ("%d", &alunos);
```

```
#define TAMANHO   100

main( )
{
    int     quantidade,  media = 0;
    float         notas [ TAMANHO ];

    // quantidade deve ser  ≤  TAMANHO
    printf ( "quantas notas devo ler ?");
    scanf("%d", &quantidade);

    for ( i = 0; i < quantidade; i++) {
      printf ( "entre com a nota %d", i+1);
```

```
    scanf("%d", &notas [ i ]);
  }
    :    :    :
  for ( i = 0; i < quantidade; i++)
    media += notas [ i ];
    :    :    :
}
```

Verificando limites

- ✓ C não realiza verificação de limites em arranjos
- ✓ nada impede o acesso além do fim do arranjo

RESULTADOS IMPREVISÍVEIS

- ✓ faça sempre que necessário a verificação dos limites

Ex:

```
#define TAM 100
    int  notas [ TAM ],  quantidade;
    :    :    :
    do  {
      printf ( "quantas notas devo ler ?");
      scanf("%d", &quantidade);
    } while ( quantidade > TAM );
```

Dimensionando um arranjo

- ✓ é possível inicializar um arranjo sem que se defina a sua dimensão
- ✓ indica-se os inicializadores, o compilador fixará a dimensão do arranjo

Ex:

```
int   notas[ ] = { 0, 0, 1, 3 }
=
int   notas[ 4 ] = { 0, 0, 1, 3 }
```

Obs: Você não pode inicializar o i-ésimo elemento sem
inicializar todos os anteriores

```
int notas [ 5 ]  = { , , 0, , }  //  ERRO
int notas [    ]  = { , , 0, , }  //  ERRO

int notas [ 5 ] =  {1, 2 }
=
int notas [ 5 ] =  {1, 2, 0, 0, 0 }
```

Arranjos Multidimensional

Ler a nota de todos os alunos do 3o. ASI

Solução: int mat1[40], mat2[40], ... mat5[40];

Problema: tratar cada variável (vetor) individualmente

Ex:

```
printf (“entre com as notas de Ltp1 \n”);
for (i = 0; i < 40; i++) {
    printf (“\n  entre com a nota %d ”, i+1);
    scanf (“%d”, &mat1[ i ]);
}
      :     :     :
printf (“entre com as notas de Inglês \n”);
for (i = 0; i < 40; i++) {
    printf (“\n  entre com a nota %d ”, i+1);
    scanf (“%d”, &mat5[ i ]);
}
```

✓ em ‘C’ podemos definir um vetor em que cada posição temos um outro vetor (matriz). Note:

int matéria [4] [40];

✓ interpretação:
temos 4 matérias, cada uma com 40 alunos

✓ Agora temos:

```
int    i,  j, matéria [ 4 ] [ 40 ];
```

```
for ( i = 0 ; i < 4; i++ ) {
    printf (“entre com as notas da matéria %d”, i+1);
    for ( j = 0; j < 40; j++) {
  printf (“entre com a nota do aluno %d”, j+1);
  scanf (“%d”, &materia [ i ] [ j ]);
    }
}
```

Inicalizando Matrizes

dado:

```
#define linhas 3
#define colunas 4

int   nota [ linhas ] [ colunas ];
```

podemos:

✓ int nota [3] [4] = { {0, 0, 0, 0}, ..., {0, 0, 0, 0 } }

✓ **nota [0] [0] = 0; ... nota [0] [3] = 0;**
: : :
✓ **nota [2] [0] = 0; ... nota [2] [3] = 0;**

✓ usar laços for

```
for  ( i = 0;   i < linhas;  i++ )
    for ( j = 0;   j < colunas;  j++ )
        nota [ i ]  [ j ]  =  0;
```

String

- ✓ é uma sequência de caracter delimitada por aspas duplas

Ex:

```
printf ( “Isto e um teste” );
printf ( “%s”, “Isto e um teste” );
```

Obs:

- ✓ visto pelo compilador: **“Isto e um teste\0”**
- ✓ ‘\0’ (null) ≠ ‘0’
- ✓ ‘\0’ indica para as funções o fim de um string

Variável String

- ✓ matriz do tipo char terminada pelo caractere null ‘\0’
- ✓ cada caractere de um string pode ser acessado individualmente

Ex:

```
char string[10] = “exemplo” ;
char string[10] = { “exemplo” };
```

```
char  string[10]  = { 'e', 'x', 'e', 'm', 'p', 'l', 'o', '\0' };

printf ( "%s", string );
printf ( "%c", string [ 0 ] );
```

Obs:

vetor de tamanho n → string de tamanho (n - 1)

Lendo Strings

- ✓ scanf

  lê o string até que um branco seja encontrado

Ex:

```
main ( )
{
  char  nome[40];

  printf ( "Digite seu nome: " );
  scanf ( "%s", &nome[ 0 ] );
  printf ( "Bom dia %s", nome );
}
```

Saida:

Digite seu nome: **Jose Maria**
Bom dia **Jose**

✓ gets
lê caracteres até encontrar '\n'
substitui '\n' por '\0'

Ex:

```
main ( )
{
 char  nome[40];
 printf ( "Digite seu nome: " );
 gets ( &nome[ 0 ] );
 printf ( "Bom dia %s", nome );
}
```

Saida:
Digite seu nome: **Jose Maria**
Bom dia **Jose Maria**

Imprimindo Strings

✓ printf
✓ puts
complemento de gets

Ex:

```
main ( )
{
 char  nome[40];
 printf ( "Digite seu nome: " );
 gets ( &nome[ 0 ] );
 puts ( "Bom dia " );
 puts ( nome );
```

```
}
```

Saida:

Digite seu nome: **Jose Maria**
Bom dia
Jose Maria

Lembre-se

✓ Sempre que uma função espera receber um apontador podemos passar:

o endereço da primeira posição do vetor/matriz

o próprio vetor/matriz

Obs: desde que o tipo seja o mesmo

Ex:

```
char  nome[40];
gets ( &nome[ 0 ] )   =   gets ( nome )

scanf (“%s”, nome[ 0 ] )   =  scanf (“%s”, nome[ 0 ] )
puts (&nome [ 0 ] )  =  puts ( nome )
```

NOTE...

```
main ( )
{
    char  nome[40];
```

```
    printf ( “Digite seu nome: “ );
    gets ( &nome[ 0 ] );
    printf ( “%s \n”, &nome[ 3 ] );
    printf ( “%s \n”, &nome[ 0 ] );
    printf ( “%s \n”, nome );
}
```

Saida:

Digite seu nome: **Jose Maria**

e Maria
Jose Maria
Jose Maria

Funções de Manipulação de Strings

✓ strlen

retorna o tamanho do string - não conta ‘\0’

Ex:

```
main ( )
{
  char  nome[40];
  printf ( “Digite seu nome: “ );
  gets ( &nome[ 0 ] );
  printf (“Tamanho = %d”, strlen(&nome[ 0 ]) );
}
```

Saida:

Digite seu nome: **Jose Maria**
Tamanho = 10

✓ strcat (str1, str2)
concatena **str2** ao final de **str1**

Ex:

```
main ( )
{
 char      nome[40]  = “Jose”,
      sobrenome[30] =  “Maria”;

 strcat(nome, sobrenome);
 puts (sobrenome);
 puts (nome);
}
```

Saida:

```
Maria
JoseMaria
```

Cuidado:
dado **str1 + str2** tem que caber em str1

✓ strcmp (str1, str2)

compara dois strings retornando
- negativo se str1 < str2
- 0 se str1 = str2
- positivo se str1 > str2

✓ a compração é feita por ordem alfabética

```
main ( )
{
  char       nome[40]  = “Jose”,
             sobrenome[30] =  “Maria”;

  if  ( strcmp ( nome, sobrenome ) )
      puts ( “os strings são diferentes” );
  else
      puts ( “os strings são identicos” );
}
```