

# Estruturas de Dados

- Prof. Edkallenn Lima
- [edkallenn@yahoo.com.br](mailto:edkallenn@yahoo.com.br) (somente para dúvidas e **NÃO PARA EXERCÍCIOS!**)

- **Blogs:**

- <http://professored.wordpress.com> (Computador de Papel – O conteúdo da forma)
- <http://professored.tumblr.com/> (Pensamentos Incompletos)
- <http://umcientistaporquinzena.tumblr.com/> (Um cientista por quinzena)
- <http://eulinoslivros.tumblr.com/> (Eu Li nos Livros)
- <http://linabiblia.tumblr.com/> (Eu Li na Bíblia)

- **Redes Sociais:**

- <http://www.facebook.com/edkallenn>
- <http://twitter.com/edkallenn>
- <https://plus.google.com/u/0/113248995006035389558/posts>
- [Pinterest: https://www.pinterest.com/edkallenn/](https://www.pinterest.com/edkallenn/)
- [Instagram: http://instagram.com/edkallenn ou @edkallenn](http://instagram.com/edkallenn)
- [LinkedIn: br.linkedin.com/in/Edkallenn](http://br.linkedin.com/in/Edkallenn)
- [Foursquare: https://pt.foursquare.com/edkallenn](https://pt.foursquare.com/edkallenn)

- **Telefones:**

- 68 98401-2103 (VIVO) e 68 3212-1211.

- Os exercícios devem ser enviados **SEMPRE** para o e-mail: [edkevan@gmail.com](mailto:edkevan@gmail.com) ou para o e-mail: [edkallenn.lima@uninorteac.edu.br](mailto:edkallenn.lima@uninorteac.edu.br)



A linguagem é o único  
instrumento da ciência.

- Samuel Johnson



# Estudo de C

- Portanto, o que faz esse programa?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 main() {
6
7     int x, y, z;
8
9     x = 10;
10    y=0;
11    z=5;
12
13    printf("\nEu vou tirar %d em Estruturas de Dados\n", x);
14    printf("\nEu nao vou tirar nem %d nem %d em Estruturas de Dados\n\n", y,z);
15
16    system("pause");
17
18
19
20
21 }
22
```



# Comentários

- Os comentários na Linguagem C começam com **/\*** e terminam com **\*/**
- Comentários **documentam** programas e melhoram sua **legibilidade**
- C99 também aceita comentários de linha única de C++, que começam com **//**
- Comentários não geram nenhuma ação por parte do computador quando o programa é executado
- Eles são ignorados pelo compilador C e não geram a criação de nenhum código-objeto em linguagem de máquina



# Exemplos:

```
re X *exemplo01.c X
1 #include <stdio.h>
2 #include <stdlib.h>
3 // #include <math.h>      // necessário para sin()
4
5 int main()
6 {
7     float y; /* Declarando a variável y */
8     y = sin(1.5);
9     printf("y = %f\n\n", y);
10    printf(""); // comentários de uma linha são permitidos
11    system("pause");
12    return 0;
13 }
14 /* O comentário pode conter mais de uma linha
15 como neste exemplo. Comentários não tem ação
16 ou não geram nenhuma ação por parte do compu-
17 tador. Eles são ignorados pelo compilador.
18 Servem para melhorar a legibilidade de programas
19 */
20
```

# Estudo de C - Entrada e Saída

- Função “printf”:
- A função printf() é uma das funções de saída/escrita de dados da linguagem C
- Seu nome vem da expressão em inglês **print formatted**, ou seja, saída (impressão) formatada
- Basicamente, a função printf() escreve na saída de vídeo (tela) um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o formato especificado
- Sua sintaxe é:

printf(“tipos de saída”, lista de variáveis);

- “tipos de saída”: conjunto de caracteres que especifica o formato dos dados a serem escritos e/ou o texto a ser escrito.
- “lista de variáveis”: conjunto de nomes de variáveis, separados por vírgula, que serão escritos



# printf() – escrevendo texto

- A função printf() pode ser usada quando queremos escrever apenas uma mensagem de texto simples na tela:

## Exemplo: escrevendo um texto na tela

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(){
04
05     printf("Esse texto sera escrito na tela");
06     system("pause");
07     return 0;
08 }
```

Saída	Esse texto sera escrito na tela
-------	---------------------------------

- O texto a ser escrito deve ser sempre definido entre **aspas duplas**



# Printf() – escrevendo valores formatados

- Quando queremos escrever dados formatados na tela usamos a sintaxe da função que possui os “tipos de saída”
- Eles especificam o formato de saída dos dados que serão escritos pela função printf().
- Cada tipo de saída é precedido por um sinal de %, e um tipo de saída deve ser especificado para cada variável a ser escrita.
- Assim, se quiséssemos escrever uma única expressão com o comando printf(), faríamos



```
← printf("%tipo_de_saída", expressão);
```

- Se fossem duas as expressões a serem escritas:



```
← printf("%tipo1 %tipo2", expressão1, expressão2);
```



# Junto ao tipo de saída pode-se adicionar texto e não apenas espaços

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     int x = 10;
7     printf("Total = %d\n",x);
8     printf("%d caixas\n",x);
9     printf("Total de %d caixas\n",x);
10    system("pause");
11    return 0;
12}
13 }
```

- Saída:
- Total = 10
- 10 caixas
- Total de 10 caixas



# Estudo de C – Exemplos - printf

```
printf("%d %g", 33, 5.3);
```

tem como resultado a impressão da linha:

33 5.3

```
printf("Inteiro = %d Real = %g", 33, 5.3);
```

com saída:

Inteiro = 33 Real = 5.3



# Exemplo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     int x = 10;
7     //Escrita de um valor inteiro
8     printf("%d\n",x);
9     float y = 5.0;
10    //Escrita de um valor inteiro e outro real
11    printf("%d%f\n",x,y);
12    //Adicionando espaço entre os valores
13    printf("%d %f\n",x,y);
14    system("pause");
15    return 0;
16
17 }
```

- Saída:
- 10
- 105.000000
- 10 5.000000



# E/S – printf() – Especificação de formato

Código	Formato
%c	<b>Caractere</b>
%d	<b>Inteiros decimais com sinal</b>
%i	Inteiros decimais com sinal
%e	Notação Científica (e minúsculo)
%E	Notação Científica (E maiúsculo)
%f	<b>Ponto flutuante decimal</b>
%g	<b>Usa %e ou %f, o que for mais curto</b>
%G	<b>Usa %E ou %F, o que for mais curto</b>
%o	Octal sem sinal
%s	<b>String de caracteres</b>
%u	Inteiros decimais sem sinal
%x	Hexadecimal sem sinal (Letras minúsculas)
%X	Hexadecimal sem sinal (letras maiúsculas)
%p	<b>Apresenta um ponteiro</b>
%%	Escreve o símbolo %



# Estudo— E/S

- Principais (mais comuns) constantes “barra invertida”:
  - \n - caractere de nova linha
  - \t - caractere de tabulação
  - \r - caractere de retrocesso
  - \" - caractere “
  - \\ - caractere \



# Estudo de C – Revisando tags do printf()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 main()
5 {
6     char str;
7     int n1;
8     float x;
9     double xx;
10
11     // Atribuindo valor as variaveis
12
13     str='a';
14     n1=234;
15     x=2.5;
16     xx=2.45;
17
18     printf("str = %c\n", str);    //caracteres %c
19     printf("n1 = %d\n", n1);    // inteiros %d
20     printf("x = %f\n", x);      // float %f
21     printf("xx = %g\n", xx);    // double pode ser %lf ou %G ou %g ou %e ou %E
22
23     system("pause");
24
25
26 }
```



# Exemplos

```
1 #include "stdio.h"
2
3 // Função : Alguns exemplos de utilizacao da funcao printf
4 // Autor : Edkallenn
5 // Data : 06/04/2012
6 // Observações:
7
8 main()
9 {
10     unsigned num;
11
12     printf ("DEC\tOCTAL\tHEX-min\tHEX-MAI\n");
13
14     for (num=0; num<255; num++) {
15         printf ("%d\t", num);
16         printf ("%o\t", num);
17         printf ("%x\t ", num);
18         printf ("%X\n", num);
19
20     }
21 }
22 }
```

# Exemplos printf

```
1 #include "stdio.h"
2
3 // Função : Alguns exemplos de utilizacao da funcao printf
4 // Autor : Edkallenn
5 // Data : 06/04/2012
6 // Observações:
7
8 main()
9 {
10     int exemplo;
11
12     printf("\n\n%p\n", exemplo); // %p mostra o endereco de maquina em um formato
13     exemplo++;                // compativel com o tipo de endereçamento
14     printf("%p\n", exemplo);   // utilizado pelo computador
15                                // Experimente trocar int exemplo por int *exemplo
16 }
17
```

```
1 #include <stdio.h>
2
3 // Função : Alguns exemplos de utilizacao da funcao
4 // Autor : Edkallenn
5 // Data : 06/04/2012
6 // Observações:
7
8 main()
9 {
10     printf("Justificando a direita:%8d\n", 100);
11     printf("Justificando a esquerda: %-8d\n", 100);
12
13 }
14
```



# Exemplos printf()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 // Função : Alguns exemplos de utilizacao da funcao printf
4 // Autor : Edkallenn
5 // Data : 06/04/2012
6 // Observações:
7 main()
8 {
9     double item = 10.12304;
10
11    printf("%f\n", item);
12    printf("%10f\n", item);
13    printf("%012f\n", item);
14    printf("%010.2f\n\n", item);
15
16    printf("%10.15s\n\n", "Esse e um teste simples");
17    //Em strings, o especificador de precisao determina o comprimento minimo e
18    //maximo do campo. Ex. %5.7s mostra uma string de pelo menos 5 e nao
19    //excedendo sete caracteres.
20    //Aplicado a inteiros o identificador de precisao determina o numero minimo
21    //de digitos que aparecerao para cada numero. Zeros iniciais seraо adiciona-
22    //dos para completar o numero solicitado de digitos
23    //Exemplo:
24    printf("%9.8d\n", 1000);
25    printf("%.3f\n", item);
26    system("pause");
27 }
```

# Estudo de C – E/S

- Função “**scanf**”:
- captura valores fornecidos via teclado

`scanf ("tipos de entrada", lista de variáveis...);`

- Ex:

```
int n;  
scanf ("%d", &n);
```

- valor inteiro digitado pelo usuário é armazenado na variável n



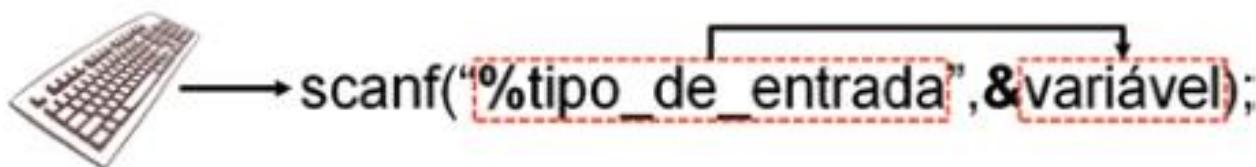
# scanf()

- A função scanf() recebe dois parâmetros de entrada:
- “tipos de entrada” → : conjunto de caracteres que especifica o formato dos dados a serem lidos.
- lista de variáveis → : conjunto de nomes de variáveis que serão lidos e separados por vírgula, em que cada nome de variável é precedido pelo operador &.

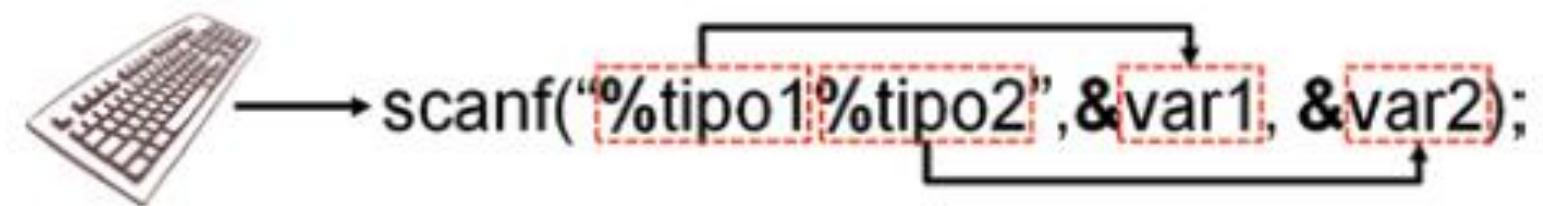


# scanf()

- Os tipo de entrada especificam o formato de entrada dos dados que serão lidos pela função scanf().
- Cada tipo de entrada é precedido por um sinal de %, e um tipo de entrada deve ser especificado para cada variável a ser lida. Assim, se quiséssemos ler uma única variável com o comando scanf(), faríamos:



- Se fossem duas variáveis, faríamos:



- E assim por diante.



# E/S – scanf() – Especificação de formato

Código	Formato
%c	Lê um único caractere
%d	Lê um Inteiro decimal
%i	Lê um Inteiro decimal
%e	Lê um número em ponto flutuante
%f	Lê um número em ponto flutuante
%g	Lê um número em ponto flutuante
%o	Lê um número octal
%s	Lê uma string
%u	Inteiros decimais sem sinal
%x	Lê um número Hexadecimal
%p	Lê um um ponteiro
%u	Lê um inteiro sem sinal
%[ ]	Busca por um conjunto de caracteres



# Exemplo scanf()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5
6     int x,z;
7     float y;
8     //Leitura de um valor inteiro
9     scanf("%d",&x);
10    //Leitura de um valor real
11    scanf("%f",&y);
12    //Leitura de um valor inteiro e outro real
13    scanf("%d%f",&x,&y);
14    //Leitura de dois valores inteiros
15    scanf("%d%d",&x,&z);
16    //Leitura de dois valores inteiros com espaço
17    scanf("%d %d",&x,&z);
18    system("pause");
19    return 0;
20
21 }
```



# scanf()

- No exemplo anterior, os comandos

```
scanf("%d%d", &x, &z);
```

e

```
scanf("%d %d", &x, &z);
```

- são **equivalentes**.
- Isso ocorre porque o comando scanf() **ignora os espaços em branco** entre os tipos de entrada.
- Além disso, quando o comando scanf() é usado para ler dois ou mais valores, podemos optar por duas formas de digitar os dados no teclado:
  - Digitar um valor e, em seguida, pressionar a tecla ENTER. Fazer isso para cada valor a ser digitado.
  - Digitar todos os valores separados por espaço e, por último, pressionar a tecla ENTER.



# Estudo de C – E/S

- Função “**scanf**” (cont.):
- caracteres diferentes dos especificadores no formato servem para cercar a entrada
- espaço em branco dentro do formato faz com que sejam "pulados" eventuais brancos da entrada
- %d, %f, %e e %g automaticamente pulam os brancos que precederem os valores numéricos a serem capturados

```
scanf ("%d:%d", &h, &m);
```

```
/* valores (inteiros) fornecidos devem ser  
separados pelo caractere dois pontos (:) -  
hora */
```



# Estudo de C - Operadores

- A atribuição “`=`” é um operador
- Operadores aritméticos (`+`, `-`, `*`, `/`, `%`, `--` e `++`):
- Operações são feitas na precisão dos operandos
- O operando com tipo de menor expressividade é convertido para o tipo do operando com tipo de maior expressividade (casting – será estudado com mais detalhes)
- ATENÇÃO: Divisão entre inteiros trunca a parte fracionária

```
int a;
double b, c;
a = 3.5;                      /* a recebe o valor 3 */
b = a / 2.0;                   /* b recebe o valor 1.5 */
c = 1/3 + b;                   /* 1/3 retorna 0 pois a operação
                                será sobre inteiros */
c = b;                         // c recebe o valor de b
```



# Avaliação de expressões aritméticas

- Os operadores (binários) aritméticos disponíveis na linguagem C são:

Operador	Operação
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão



# Operador, significado e exemplo

Operador	Significado	Exemplo
+	adição de dois valores	$z = x + y$
-	subtração de dois valores	$z = x - y$
*	multiplicação de dois valores	$z = x * y$
/	quociente de dois valores	$z = x / y$
%	resto de uma divisão	$z = x \% y$



# Prioridade (aritméticas)

- **Prioridade de execução das operações:**
  - 1) expressões entre parênteses
  - 2) multiplicação, divisão e resto da divisão (da esquerda para a direita) – operadores multiplicativos
  - 3) operações de soma e subtração (da esquerda para a direita) – operadores aditivos



# Execute este programa:

```
1 #include <stdio.h>
2 /* Funcao: Programa exemplo do operador de atribuicao
3    Autor: Edkallenn
4    Data: 06/04/2012
5    Observacoes: Programa esta no exemplo do slide.
6 */
7
8 main() {
9     //declaracao
10    int a;
11    double b, c;
12
13    a=3.5;           // a recebe o valor 3
14    b= a / 2.0;      // b recebe o valor 1.5
15    c= (1/3) + b;   // 1/3 retorna 0 pois a operacao sera
16    c=b;            //sobre inteiros - c recebe b
17
18    printf("%d - %G - %f\n\n", a, b, c);
19 }
```



# Estudo de C - Operadores

- O operador módulo, “%”, aplica-se a inteiros
- precedência dos operadores: \* / - +
- Ex:

**x % 2**

*/\* o resultado será 0, se x for par;  
caso contrário, será 1 \*/*

**a + b \* c / d** //é equivalente a  
**(a + ((b \* c) / d))**

- Depois veremos a tabela completa de precedência



# Operadores de atribuição

= , += , -= , \*= , /= , %=

- Notação compacta para atribuições em que a mesma variável aparece dos dois lados.

**var op= expr**

é equivalente a

**var = var op (expr)**

- **i += 2;** é equivalente a **i = i + 2;**
- **x \*= y + 1;** é equivalente a **x = x \* (y +1);**



# Operadores de atribuição Compactos

- RESUMO

Operador	Significado	Exemplo		
<code>+=</code>	soma e atribui	<code>x += y</code>	igual	<code>x = x + y</code>
<code>-=</code>	subtrai e atribui	<code>x -= y</code>	igual	<code>x = x - y</code>
<code>*=</code>	multiplica e atribui	<code>x *= y</code>	igual	<code>x = x * y</code>
<code>/=</code>	divide e atribui quociente	<code>x /= y</code>	igual	<code>x = x / y</code>
<code>%=</code>	divide e atribui resto	<code>x %= y</code>	igual	<code>x = x % y</code>



# Estudo de C - Operadores

- Operadores de incremento e decremento ( **++** , **--** ):
  - Incrementa ou decremente de uma unidade o valor de uma variável
    - Os operadores não se aplicam a expressões
    - O incremento pode ser antes ou depois da variável ser utilizada
- **n++** incrementa n de uma unidade, depois de ser usado
- **++n** incrementa n de uma unidade, antes de ser usado

```
n = 5;  
x = n++;      /* x recebe 5; n é incrementada para 6 */  
x = ++n;      /* n é incrementada para 6; x recebe 6 */  
a = 3;  
b = a++ * 2;  
  
/* b termina com o valor 6 e a com o valor 4 */
```





## Operações sem Dor

### Nem todos sinais de igualdade são iguais.

Em C, o sinal de igualdade (`=`) é usado para **atribuição**. Mas um sinal duplo de igualdade (`==`) é usado para **verificar igualdade**.

`teeth`  
recebe o valor 4.  
 $\rightarrow$  `teeth = 4;`

`teeth == 4;`  
 $\uparrow$   
 Verifica se o valor de `teeth` é igual a 4.

Se quiser aumentar ou diminuir uma variável, você pode economizar espaço com as atribuições `+=` e `-=`.

Soma 2 a `teeth`.  
 $\downarrow$   
`teeth += 2;`

`teeth -= 2;`

Diminui 2 de `teeth`.  
 $\uparrow$

Finalmente, se quiser incrementar ou decrementar uma variável em um, use `++` e `--`.

`teeth++;`  $\leftarrow$  Incrementa em um.

`teeth--;`  $\leftarrow$  Decrementa em um.



# Estudo de C - Operadores

- Operadores relacionais  
 $(<, \leq, ==, \geq, >, !=)$ :
- Relacional refere-se às relações que os valores podem ter uns com os outros.
- O resultado será 0 ou 1 (não há valores intrinsecamente booleanos em C – como em Java, por exemplo)

```
int a, b;  
int c = 23;  
int d = c + 4;  
c < 20 retorna 0  
d > c retorna 1
```



# Estudo de C - Operadores

- Operadores relacionais - RESUMO

Operador	Significado	Exemplo
>	Maior do que	$x > 5$
$\geq$	Maior ou igual a	$x \geq 10$
<	Menor do que	$x < 5$
$\leq$	Menor ou igual a	$x \leq 10$
$\equiv$	Igual a	$x \equiv 0$
$\neq$	Diferente de	$x \neq 0$



# Estudo de C - Operadores

- Operadores lógicos ( **&&** , **||** , **!** )
  - Lógico refere-se à maneira como as relações entre operadores podem ser conectadas
  - A avaliação é da esquerda para a direita
  - A avaliação para quando o resultado pode ser conhecido

```
int a, b;  
int c = 23;  
int d = c + 4;
```

```
a = (c < 20) || (d > c);  
/* retorna 1 - as duas sub-expressões são avaliadas */
```

```
b = (c < 20) && (d > c);  
/* retorna 0 - apenas a primeira sub-expressão é avaliada */
```



# Tabela verdade de operadores lógicos

p	q	p&&q	p  q	!p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

- Os operadores relacionais e lógicos são menores em precedência do que os operadores aritméticos
- Uma expressão como  $10 > 1 + 12$  é avaliada como se fosse escrita  $10 > (1 + 12)$ . O resultado é, obviamente, **falso**.
- Pode se combinar diversas operações em uma expressão, como:  
 $10 > 5 \&\& !(10 < 9) || 3 <= 4$ . Nesse caso o resultado é?
- C não tem nativamente o operador lógico xor. O resultado de uma operação XOR é verdadeiro, se, e somente se, um operando mas não os dois for verdadeiro.
- **DESAFIO:** Construa uma função que retorne o resultado de uma operação XOR – enviar para o email do professor.



# Operadores Lógicos

- RESUMO:

Operador	Significado	Exemplo
<code>&amp;&amp;</code>	Operador <b>E</b>	<code>(x &gt;= 0 &amp;&amp; x &lt;= 9)</code>
<code>  </code>	Operador <b>OU</b>	<code>(a == 'F'    b != 32)</code>
<code>!</code>	Operador <b>NEGAÇÃO</b>	<code>!(x == 10)</code>

O padrão ANSI C não tem valor para verdadeiro ou falso. Programas C tratam o valor 0 como falso e qualquer outro valor como verdadeiro. O padrão C99 permite o uso das palavras *true* e *false* nos programas – mas o compilador os trata como os valores 1 e 0 de qualquer forma.



# Precedência relativa dos operadores relacionais e lógicos

- Maior

!

>      >=      <      <=

==      !=

&&

||

- Menor

- É possível usar parênteses para alterar a ordem de precedência natural de avaliação de uma expressão



# Operador sizeof

- Retorna o tamanho, em bytes, da variável ou especificador de tipo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 /*
4 *      Função : Programa para mostrar o tamanho dos tipos em C
5 *      Autor : Edkallenn
6 *      Data : 06/04/2012
7 *      Observações: sizeof é um operador em tempo de compilação unário que
8 *                  retorna o tamanho, em bytes, da variável ou especificador
9 *                  de tipo, em parenteses que ele precede.
10 */
11 main()
12 {
13
14     printf("Tamanho do tipo de dado int: %d bytes\n", sizeof(int));
15     printf("Tamanho do tipo de dado char: %d byte\n", sizeof(char));
16     printf("Tamanho do tipo de dado float: %d bytes\n", sizeof(float));
17     printf("Tamanho do tipo de dado double: %d bytes\n", sizeof(double));
18
19     system("pause");
20
21
22 }
```

# Demais operadores

- Os operadores

\* & , . -> () [] ?  
  >> << & | ^ ~

serão estudados conforme seu uso for necessário.



# Operadores bit a bit

- **RESUMO**

Operador	Significado	Exemplo
<code>~</code>	complemento bit a bit	<code>~x</code>
<code>&amp;</code>	E bit a bit	<code>x &amp; 167</code>
<code> </code>	OU bit a bit	<code>x   129</code>
<code>^</code>	OU exclusivo	<code>x ^ 167</code>
<code>&lt;&lt;</code>	deslocamento de bits à esquerda	<code>x &lt;&lt; 2</code>
<code>&gt;&gt;</code>	deslocamento de bits à direita	<code>x &gt;&gt; 2</code>



# Resumo das precedências

- Maior

( ) [ ] ->  
! ~ ++ -- (tipo) \* & sizeof  
\* / %  
+ -  
<< >>  
< <= > >=  
== !=  
&  
^  
|  
&&  
||  
?:  
= += -= \*= /= etc.

- Menor



# Exemplos de scanf()

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 // Função : Converte altura em metros para pés e polegadas
4 // Autor : Waldemar Celes
5 // Data : 06/04/2012
6 // Observações: Demonstra o uso de scanf
7
8 main()
9 {
10     int pes;                      //numero de pes
11     float polegadas,altura; //polegadas e altura em metros
12
13     //Captura a altura em metros
14     printf("Digite a altura em metros: ");
15     scanf("%f", &altura);
16
17     //calcula a altura em pés e polegadas
18     altura = 100 * altura;          //converte em centimetros
19     pes = (int) (altura/30.48);    //calcula o numero de pes
20     polegadas = (altura-pes * 30.48) / 2.54 ;
21
22     //exibe a altura convertida
23     printf("Altura: %d pes %.1f polegadas\n", pes, polegadas);
24     system("pause");
25 }
```

Desenvolva os algoritmos e codificação em linguagem C dos seguintes programas. Não se esqueça de ir gravando cada programa.

- Efetuar o cálculo da quantidade de litros de combustível gasta em uma viagem, utilizando um automóvel que faz 12 km por litro. Para obter o cálculo , o usuário deve fornecer **o tempo** gasto na viagem e a **velocidade média**. Desta forma, será possível obter a **distância** percorrida com a fórmula  $\text{DISTÂNCIA} = \text{TEMPO} * \text{VELOCIDADE}$ . Tendo o valor da distância, basta calcular a **quantidade de litros** de combustível utilizada na viagem com a fórmula:  $\text{LITROS\_USADOS} = \text{DISTÂNCIA} / 12$ . O programa deve apresentar os valores da **velocidade média**, **tempo gasto**, a **distância percorrida** e a **quantidade de litros** utilizada na viagem. Dica: Trabalhe com valores reais. Em seguida refaça o programa para receber a autonomia (a quantidade de km por litro).
- Ler uma **temperatura em graus Celsius** e apresentá-la convertida em graus **Fahrenheit**. A fórmula de conversão de temperatura a ser utilizada é  $F = (9 * C + 160) / 5$ , em que a variável **F** representa é a temperatura em graus **Fahrenheit** e a variável **C** representa é a temperatura em graus **Celsius**.



- Ler uma temperatura em graus **Fahrenheit** e apresenta-la convertida em graus **Celsius**. A fórmula de conversão de temperatura a ser utilizada é  $C = (F - 32) * 5 / 9$ , em que a variável **F** é a temperatura em graus **Fahrenheit** e a variável **C** é a temperatura em graus **Celsius**.
- Calcular e apresentar o valor do **volume** de uma **lata de óleo**, utilizando a fórmula:  
 $V = 3.14159 * \text{raio} * \text{raio} * \text{altura}$ , em que as variáveis: **V**, **raio** e **altura** representam respectivamente o **volume**, o **raio** e a **altura**.
- Ler dois valores **inteiros** para as variáveis **A** e **B**, **efetuar a troca dos valores** de modo que a variável **A** passe a possuir o valor da variável **B**, e a variável **B** passe a possuir o valor da variável **A**.  
**Apresentar os valores trocados**



# Estudo de C – Exercícios

- Antes, vamos praticar um pouco:
  1. Faça um programa para ler um número real e exibir o seu cubo, o seu dobro, o seu quádruplo, o seu quíntuplo e a sua raiz quadrada.
  2. Ler dois números inteiros e exibir o resto da divisão do primeiro pelo segundo.
  3. Ler dois números inteiros e escrever o valor de seus quadrados e de suas raízes quadradas.
  4. Ler três números reais e exibir o triplo de sua soma
  5. Ler um valor em reais e exibir o equivalente em dólares. Considere \$ 1 = R\$ 1,83.
  6. Ler o salário de um funcionário e imprimi-lo com um aumento de 12%.
  7. Ler dois valores reais e exibir o primeiro com um acréscimo de 30%, e o segundo com um desconto de 25%.
  8. Calcular e imprimir a área de um triangulo ( $ha = b * a / 2$ )
  9. Calcular a área de um triangulo Retângulo usando o Teorema de Heron onde  $area=sqrt(s(s-a)(s-b)(s-c))$  e  $s = (a+b+c)/2$ . S é o semiperímetro do triângulo.
  10. Ler a idade de uma pessoa e mostrar, aproximadamente, quantos dias de vida ela viveu (assuma que um ano tem 365 dias)
- Fazer em sala de aula TODOS os exercícios.



# Estudo de C — Controle de Fluxo

- Tomada de decisão
- Construções com laços
- Seleção



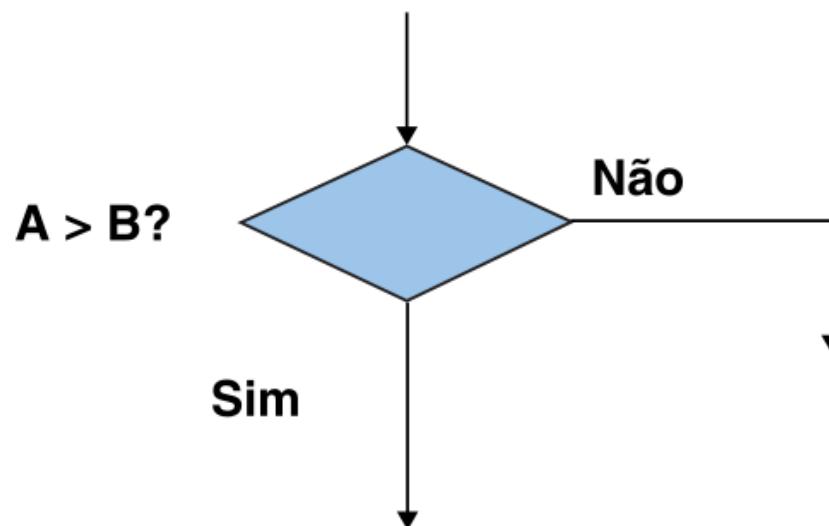
# Controle de Fluxo - Condicional

- Os programas escritos até o momento são programas **sequenciais**
- Um comando é **executado após o outro**, do começo ao fim do programa, na ordem em que foram declarados no código-fonte.
- **Nenhum comando é ignorado**
- Há casos em que é preciso que um bloco de comandos **seja executado somente se determinada condição for verdadeira**
- Precisamos de uma estrutura de **SELEÇÃO** ou um comando de controle **CONDICIONAL**



# Condicional

- Isso é muito similar ao que ocorre em um **fluxograma**, em que o símbolo do losango permite **escolher** entre diferentes **caminhos** com base em uma **condição** do tipo verdadeiro/falso



# Definindo condição

- **Condição** é qualquer expressão relacional (ou seja, que use os operadores  $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ,  $\neq$  ou  $\equiv$ ) que resulte em uma resposta do tipo verdadeiro ou falso.
- Por exemplo, para a condição  $x > 0$  temos que:
  - Se o valor de  $x$  for um valor POSITIVO, a condição será considerada verdadeira.
  - Se o valor de  $x$  for igual a ZERO ou NEGATIVO, a condição será considerada falsa.



# Expressão condicional

- Expressão condicional é qualquer expressão que resulte em uma resposta do tipo verdadeiro ou falso.
- Ela pode ser construída utilizando operadores:
  - Matemáticos : +, -, \*, /, %
  - Relacionais: >, <, >=, <=, ==, !=
  - Lógicos: &&, ||, !
- Esses operadores permitem criar condições mais complexas, como mostra o exemplo seguinte, no qual se deseja saber se a divisão de x por 2 é maior do que o valor de y menos 3:

$$x/2 > y - 3$$



# Condicionais



Uma expressão condicional pode utilizar operadores dos tipos matemático, relacional e/ou lógico.

- |    |                                      |
|----|--------------------------------------|
| 01 | x é maior ou igual a y?              |
| 02 | $x \geq y$                           |
| 03 |                                      |
| 04 | x é maior do que y+2?                |
| 05 | $x > y+2$                            |
| 06 |                                      |
| 07 | x-5 é diferente de y+3?              |
| 08 | $x-5 \neq y+3$                       |
| 09 |                                      |
| 10 | x é maior do que y e menor do que z? |
| 11 | $(x > y) \&\& (x < z)$               |



# Comando if

- O comando if é utilizado sempre que é necessário escolher entre dois caminhos dentro do programa ou quando se deseja executar um ou mais comandos que estejam sujeitos ao resultado de um teste.
- A forma geral de um comando if é:

```
if (condição) {  
    sequência de comandos;  
}
```

- Na execução do comando if a condição será avaliada e:
  - Se a condição for **verdadeira**, a sequência de comandos **será** executada.
  - Se a condição for **falsa**, a sequência de comandos **não será** executada, e o programa continuará a partir do primeiro comando seguinte ao final do comando if.

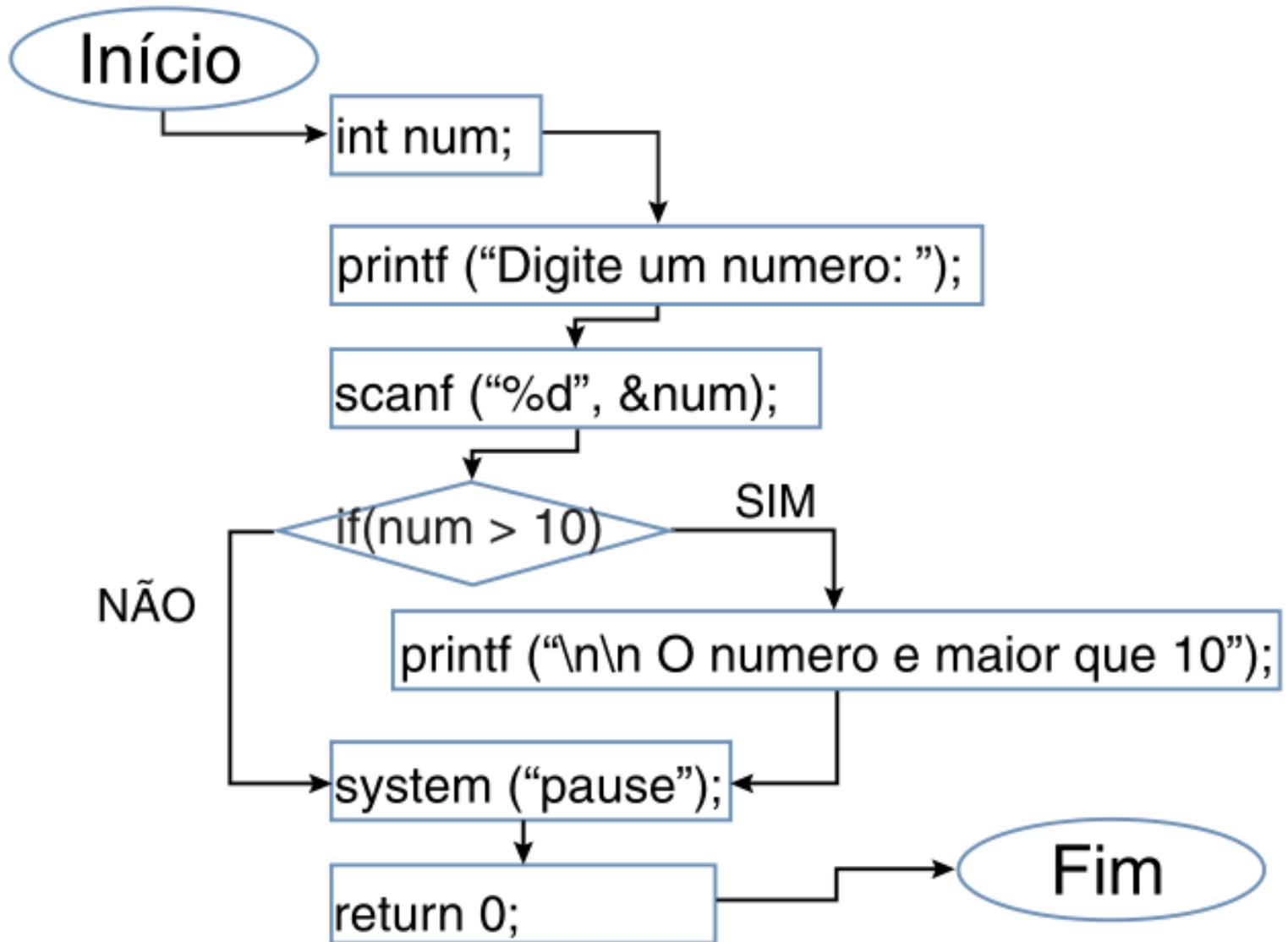


# Exemplo - Comando if

## Exemplo: comando if

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(){
04     int num;
05     printf ("Digite um numero: ");
06     scanf("%d",&num);
07     if(num > 10)
08         printf ("O numero e maior do que 10\n");
09
10    system("pause");
11    return 0;
12 }
```





# DICA



Diferentemente da maioria dos comandos, não se usa o ponto e vírgula (;) depois da condição do comando **if**.

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(){
04     int num;
05     printf ("Digite um numero: ");
06     scanf("%d",&num);
07     if(num > 10); ERRADO!
08         printf ("O numero e maior do que 10\n");
09
10     system("pause");
11     return 0;
12 }
```



# Uso das chaves

- No comando if, e em diversos outros comandos da linguagem C, usam-se os operadores de chaves ( { } ) para delimitar um bloco de instruções.
- **IMPORTANTE:** Por definição, comandos de condição (if e else) ou repetição (while, for e do while) atuam apenas sobre o comando seguinte a eles.



# Uso das chaves

- Desse modo, se o programador desejar que mais de uma instrução seja executada por aquele comando if, esse conjunto de instruções deve estar contido dentro de um bloco delimitado por chaves ({}):

```
if (condição) {  
    comando 1;  
    comando 2;  
    ...  
    comando n;  
}
```



# DICA



As chaves podem ser ignoradas se o comando contido dentro de **if** for único.

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(){
04     int num;
05     printf ("Digite um numero: ");
06     scanf("%d",&num);
07     if(num > 10)
08         printf("O numero e maior que 10\n");
09
10     /*OU
11     if(num > 10){
12         printf ("O numero e maior que 10\n");
13     }
14     */
15     system("pause");
16     return 0;
17 }
```



# O comando else

- O comando **else** pode ser entendido como um complemento do comando if.
- Ele auxilia o comando if na tarefa de **escolher** entre os **vários caminhos** a serem seguidos dentro do programa.
- O comando **else** é **opcional**, e sua sequência de comandos somente será executada se o valor da condição que está sendo testada pelo comando if for FALSA.



# else

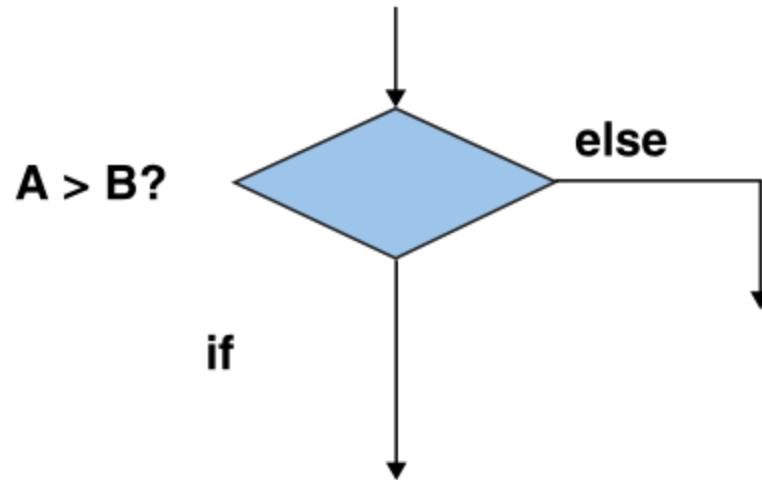
- A forma geral de um comando else é:

```
if(condição) {  
    primeira sequência de comandos;  
}  
  
else {  
    segunda sequência de comandos;  
}
```

- Se o comando **if** diz o que fazer quando a condição é verdadeira, o comando **else** trata da condição quando ela é falsa.



# Fluxograma

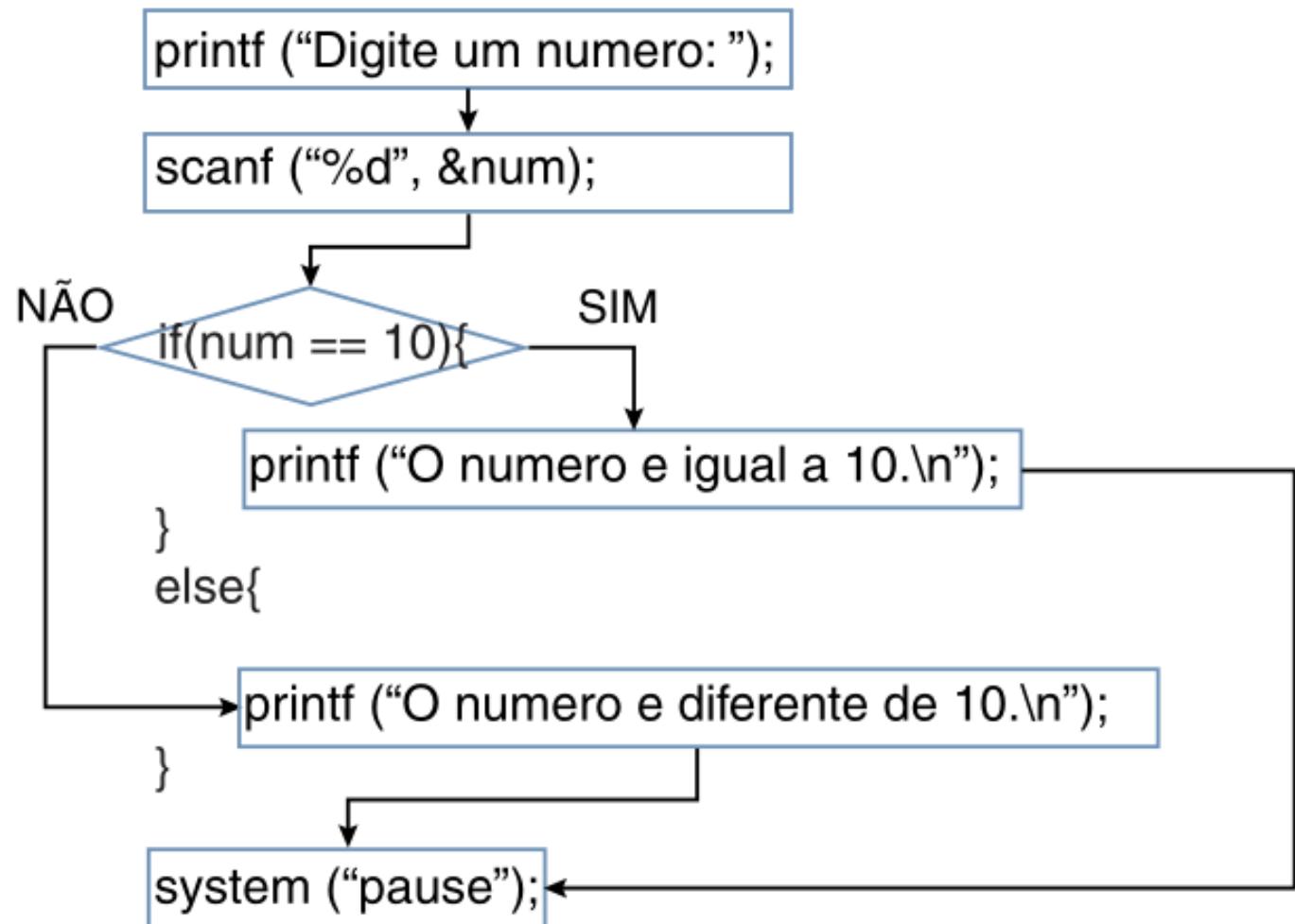


# Exemplo - else

## Exemplo: comando if-else

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(){
04     int num;
05     printf ("Digite um numero: ");
06     scanf("%d", &num);
07     if(num == 10){
08         printf ("O numero e igual a 10.\n");
09     } else{
10         printf ("O numero e diferente de 10.\n");
11     }
12     system("pause");
13     return 0;
14 }
```





# Tomada de decisão aninhada

- Exemplo:
- Programa para qualificar a temperatura:
  - Se a temperatura for menor do que 20° C, então está frio
  - Se a temperatura estiver entre 20°C e 30°C, então está agradável
  - Se a temperatura for maior do que 30°C, então está quente



# Tomada de decisão

- Comando “if”:
- Comando básico para codificar tomada de decisão
  - se *expr* for verdadeira ( $\neq 0$ ), executa o bloco de comandos 1
  - se *expr* for falsa ( $= 0$ ), executa o bloco de comandos 2

```
if ( expr )
{ bloco de comandos 1 }
else
{ bloco de comandos 2 }
```

ou

```
if ( expr )
{ bloco de comandos }
```



# Tomada de decisão

- /\* temperatura (versao 1 - incorreta) \*/
- #include <stdio.h>

```
main()
{
    int temp;

    printf("Digite a temperatura: ");
    scanf("%d", &temp);

    if (temp < 30)
        if (temp > 20)
            printf(" Temperatura agradável \n");
    else
        printf(" Temperatura quente \n");

    system("pause");
}
```



# Tomada de decisão

- /\* temperatura (versao 1 - incorreta) \*/
- #include <stdio.h>

```
main()
{
    int temp;

    printf("Digite a temperatura: ");
    scanf("%d", &temp);

    if (temp < 30)
        if (temp > 20)
            printf(" Temperatura agradável \n");
        else
            printf(" Temperatura quente \n");

    system("pause");
}
```



# Tomada de decisão

```
/* temperatura (versao 2) */  
#include <stdio.h>  
  
int main (void)  
{  
    int temp;  
    printf ( "Digite a temperatura: " );  
    scanf ( "%d", &temp );  
    if ( temp < 30 ) {  
        if ( temp > 20 )  
            printf ( " Temperatura agradável \n" );  
    }  
    else  
        printf ( " Temperatura quente \n" );  
    return 0;  
}
```



# Tomada de decisão

```
/* temperatura (versao 3) */  
#include <stdio.h>  
  
int main (void)  
{  
    int temp;  
    printf("Digite a temperatura: ");  
    scanf("%d", &temp);  
  
    if (temp < 10)  
        printf("Temperatura muito fria \n");  
    else if (temp < 20)  
        printf(" Temperatura fria \n");  
    else if (temp < 30)  
        printf("Temperatura agradável \n");  
    else  
        printf("Temperatura quente \n");  
    return 0;  
}
```



# Tomada de decisão

```
/* temperatura (versao 3) */  
#include <stdio.h>  
  
int main (void)  
{  
    int temp;  
    printf("Digite a temperatura: ");  
    scanf("%d", &temp);  
  
    if (temp < 10)  
        printf("Temperatura muito fria \n");  
    else if (temp < 20)  
        printf(" Temperatura fria \n");  
    else if (temp < 30)  
        printf("Temperatura agradável \n");  
    else printf("Temperatura quente \n");  
    return 0;  
}
```



# Tomada de decisão

- **Estrutura de bloco:**

- escopo de uma variável:
  - uma variável declarada dentro de um bloco é válida no bloco
  - após o término do bloco, a variável deixa de existir

```
if ( n > 0 )
{ int i; ... }
...           /* a variável i não existe neste ponto do programa */
```



# Tomada de decisão

- Operador Condicional (ternário)
- formato geral:

condição ? expressão1 : expressão2;

- se a condição for verdadeira, a expressão1 é avaliada; caso contrário, a expressão2 é avaliada
- Ex:

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

- Comando “if” equivalente:

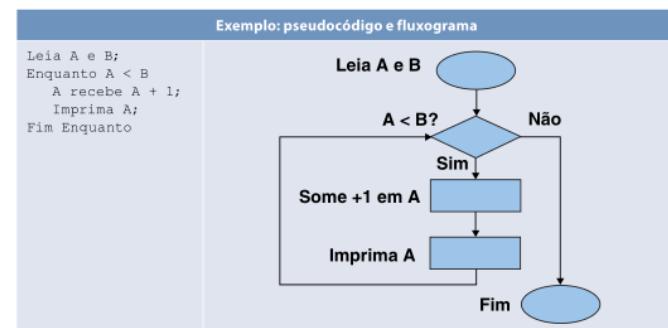
```
if ( a > b )
    maximo = a;
else maximo = b;
```



# Repetição (laços ou loops)

- Há casos em que é preciso que um bloco de comandos seja executado mais de uma vez se determinada condição for verdadeira:

enquanto **condição** faça  
 sequência de comandos;  
 fim enquanto



- Precisa-se de uma estrutura de repetição que permita executar um conjunto de comandos quantas vezes forem necessárias



# Instrução while

- A instrução while equivale ao comando “enquanto” utilizado nos pseudocódigos apresentados até agora.
- A forma geral de um comando while é:

```
while ( expr )
{
    bloco de comandos
}
```



# Instrução while

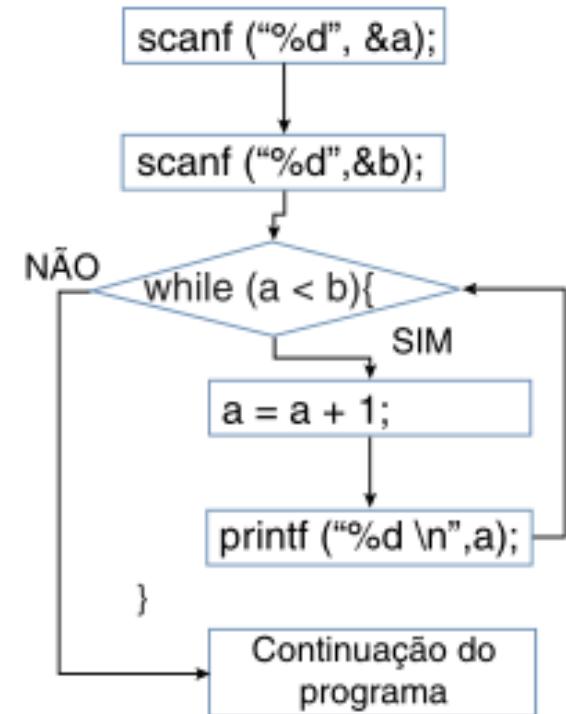
- Instrução “while” (explicação da sintaxe):
  - **enquanto** *expr* for verdadeira, o bloco de comandos é executado
  - **quando** *expr* for falsa, o comando termina (o laço se encerra)

```
while ( expr )
{
    bloco de comandos
}
```



# Exemplo anterior...

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int a,b;
6     printf("Digite o valor de a: ");
7     scanf("%d", &a);
8     printf("Digite o valor de b: ");
9     scanf("%d", &b);
10    while (a < b) {
11        a = a + 1;
12        printf("%d \n", a);
13    }
14    system("pause");
15    return 0;
16 }
```



# Loops frequentemente usam a mesma estrutura...

Você pode usar o loop `while` sempre que tiver de repetir uma parte do código, mas muitas vezes seus loops terão o mesmo tipo de estrutura:

- ★ Faça algo simples antes do loop, como determinar um contador.
- ★ Tenha uma condição de teste simples no loop.
- ★ Faça algo no fim do loop, como atualizar o contador.

# Laços – outro exemplo

- Ex: Fatorial de um número não negativo

$$n! = n \times (n - 1) \times (n - 2) \dots 3 \times 2 \times 1$$

onde:  $0! = 1$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$3! = 3 \times 2 \times 1 = 6$$

...



# Laços

- definição recursiva da função *fatorial*:
- $N \rightarrow N$

$$fatorial(0) = 1$$

$$fatorial(n) = n \times fatorial(n-1)$$

- Cálculo não recursivo de *fatorial(n)*

comece com:

$$k = 1$$

$$f = 1$$

faça enquanto  $k \leq n$

$$f = f * k$$

incremente  $k$



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 /* Funcão: Fatorial não recursivo
4 Autor: Edkallenn
5 Data: 05/09/2013
6 Observações:
7 */
8 main()
9 {
10
11     int cont, num;
12     double fat;
13     printf("\n\nDigite um numero inteiro não-negativo: ");
14     scanf("%d", &num);
15
16     fat = 1;
17     //calculo do fatorial
18     cont=1;
19     while(cont<=num) {
20         fat=fat*cont;
21         cont++;
22     }
23     printf("\n\nO fatorial de %d eh %g \n\n", num, fat);
24     system("pause");
25 }
```



# Construções com laços

- Exercício:
- Implemente o programa para calcular o fatorial (usando fat inteiro)
- Execute-o para 2, 5, 10 e 20
- Explique o programa
- O que acontece com  $n=20$ ?
- E se mudar o tipo de dados?



# Exercício de Fixação

Fazer programas para:

1. Exibir todos os números inteiros de 1 até 100
2. Exibir todos os números inteiros de 100 até 1
3. Exibir os números inteiros pares de 1 até 50
4. Exibir os números inteiros ímpares de 50 até 1
5. Exibir todos os números inteiros pares entre 1 e um número inteiro digitado pelo usuário.



# Outro exemplo

- Dado o valor da variável N, determinar a soma dos números inteiros de um a N  
(somatório com incremento 1)



# Exercício

- Faça um programa para implementar o operador somatório.
- Um **somatório** é um operador matemático que nos permite representar facilmente somas muito grandes ou até infinitas. É representado com a letra grega sigma ( $\Sigma$ ), e é definido por:

$$\sum_{i=m}^n x_i = x_m + x_{m+1} + x_{m+2} + \cdots + x_{n-1} + x_n$$



# Exercício - Somatório

- A variável  $i$  é o **índice do somatório** que designa um valor inicial chamado **limite inferior**,  $m$ .
- A variável  $i$  percorre os valores inteiros até alcançar o **limite superior**,  $n$ .
- Necessariamente tem-se  $m \leq n$
- Por exemplo se queremos expressar a soma dos dez primeiros números naturais podemos representá-lo assim:

$$\sum_{i=1}^{10} i$$

- Implemente o programa para  $\sum_{i=1}^n 2i$  e para  $\sum_{i=1}^n 3i$



```
1 #include "stdio.h"
2 #include "stdlib.h"
3
4 /*
5
6     Função : Exemplo do comando while
7     Autor : Edkallenn
8     Data : 06/04/2012
9     Observações: Somatório dos numeros inteiros de 1 a N
10
11 */
12
13 main()
14 {
15     long int N, i, soma; //N e a entrada, i eh o controle do laco
16                     // soma eh a soma dos numeros de 1 a N
17     printf("Digite o valor de N: ");
18     scanf("%d", &N);
19
20     soma = 0;    //incializa a variavel soma
21     i = 1;       //inicializa o controle do laco
22     while (i<=N)
23     {
24         soma = soma + i; //ou soma+=i
25         i++;
26     }
27
28     printf("A soma dos valores de 1 a %d eh: %d\n\n", N, soma);
29
30
31 }
```



# while

- Temos que tomar cuidado com situações de “loops infinitos”
- Este tipo de erro é causado por erros de lógica.
- Importante realizar um “esmiuçamento” do programa, **debugando**.
- O ideal é fazer um “**algoritmo**” antes para os problemas pequenos (modularizando o problema maior, transformando-os em rotinas)



# Erro (bug) – loop infinito

```
1 #include "stdio.h"
2 #include "stdlib.h"
3
4 /*
5     Função : Exemplo do comando while
6     Autor : Edkallenn
7     Data : 06/04/2012
8     Observações: Somatório dos numeros inteiros de 1 a N
9 */
10
11 main()
12 {
13     long int N, i, soma; //N e a entrada, i eh o controle do laco
14                         // soma eh a soma dos numeros de 1 a N
15     printf("Digite o valor de N: ");
16     scanf("%d", &N);
17
18     soma = 0;           //incializa a variavel soma
19     i = 0;              //inicializa o controle do laco
20     while (i < N)
21     {
22         i--;
23         soma = soma + i; //ou soma+=i
24     }
25     printf("A soma dos valores de 1 a %d eh: %d\n\n", N, soma);
26
27 }
```

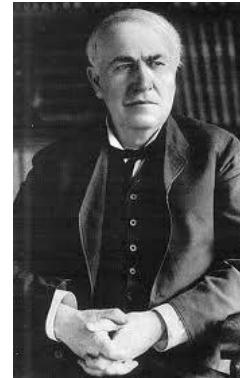


# A propósito. Bug?

- É um erro no funcionamento comum de um *software*, também chamado de falha na lógica de um programa de computador
- Gracie Marie Hopper, a maior de todas as geeks (Mark II, 1945, em Harvard) ou Thomas Edison (fonógrafo em 1878)?



x



- <http://meiobit.com/97634/grace-hopper-a-maior-de-todas-as-geeks/>

9/9

0800  
1000

arctan started

stopped - arctan ✓  
 13" sec (032) MP - MC       $\begin{cases} 1.2700 & 9.037847025 \\ 9.037846995 & \text{correct} \end{cases}$   
~~1.982147000~~ ~~2.150776715~~(-3) 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test  
 in relay " 10.000 test "

1100

Started Cosine Tape (Sine check)

1525

Started Mult + Adder Test.

1545



Relay #70 Panel F  
 (moth) in relay.

1600

First actual case of bug being found.

arctan started.

1700 closed down.

Relay  
2145  
Relay 3370

- Fotografia supostamente do primeiro Bug (um inseto real) que foi depurado ("debuggado") no MARK II (estava em um relé)



# Exercícios com laços e tomada de decisão

- Faça um programa que leia dois números e mostre qual deles é o maior. Se, por acaso, os dois números forem iguais, imprima a mensagem “Números iguais”.
- Faça um programa que leia um número e, caso ele seja positivo, calcule e mostre:
  - O número digitado ao quadrado.
  - A raiz quadrada do número digitado.
- Faça um programa que leia os coeficientes de uma equação do segundo grau. Em seguida, calcule e mostre as raízes dessa equação, lembrando que as raízes são calculadas como
$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
- em que  $\Delta = b^2 - 4 * a * c$  e  $ax^2 + bx + c = 0$  representa uma equação do segundo grau. A variável  $a$  tem de ser diferente de zero. Caso seja igual, imprima a mensagem “Não é equação de segundo grau”. Do contrário, imprima:
  - Se  $\Delta < 0$ , não existe real. Imprima a mensagem “Não existe raiz”.
  - Se  $\Delta = 0$  existe uma raiz real. Imprima a raiz e a mensagem “Raiz única”.
  - Se  $\Delta > 0$ , existem duas raízes reais. Imprima as raízes.
- Faça um programa que leia um número inteiro positivo N e imprima todos os números naturais de 0 até N em ordem crescente.
- Faça um programa que leia um número inteiro N e depois imprima os N primeiros números naturais ímpares.
- Elabore um programa que peça ao usuário para digitar 10 valores. Some esses valores e apresente o resultado na tela.
- Faça um programa que determine e mostre os cinco primeiros múltiplos de 3 considerando números maiores que 0
- Faça um programa que leia um número e exiba os 10 sucessores deste número



# Instrução “for”

- Comando “for”
- Forma compacta para exprimir “loops” ou laços

```
for (expressão_inicial; expressão_booleana; expressão_de_incremento)
{
    bloco de comandos
}
```

- Equivale a:

```
expressão_inicial;
while ( expressão_booleana )
{
    bloco de comandos
    ...
    expressão_de_incremento
}
```

here	temperatura_exemplo_if.c	soma_de_um_a_N_while.c	fatorial_nao_recursivo.c	fatorial_nao_recursivo_co
------	--------------------------	------------------------	--------------------------	---------------------------

```
1  /* Fatorial (versão 2) */
2  #include <stdio.h>
3  /*
4      Função : Fatorial não recursivo - com for
5      Autor : Edkallenn
6      Data : 06/04/2012
7      Observações:
8  */
9  int main (void)
10 {
11     int k;
12     int n;
13     int f = 1;
14     printf("Digite um número inteiro não negativo:");
15     scanf("%d", &n);
16
17     /* calcula fatorial */
18     for (k = 1; k <= n; k=k+1) { // a expressão "k = k + 1" é equivalente a "k++"
19         f = f * k; // a expressão "f = f * k" é equivalente a "f *= k" *
20     }
21     printf(" Fatorial = %d \n", f);
22     return 0;
23 }
```



# Lembrando: Transformação while - for

while

```
expressão_inicial  
while(expressão_boleana)  
{  
    bloco de comandos  
    ...  
    expressão_de_incremento  
}
```

for

```
for (expressão_inicial; expressão_boleana; expressão_de_incremento)  
{  
    bloco de comandos  
}
```

O comando **for** pode ser usado em qualquer situação em que o comando **while** poderia ser usado.

O comando **for** é muito usado em variáveis indexadas (vetor, matriz, structs e as estruturas dinâmicas).



# Exercício

- Alterar o programa do somatório para executar usando o comando for e salvá-lo com um nome diferente.
- Executar o programa para  $n=10, 100, 200$  e  $1000$  (exibindo cada iteração, junto com a soma)



# Construção “do-while”

- O teste de encerramento é avaliado no final
- Usado quando o laço deve ser repetido pelo menos uma vez.

```
do
{
    bloco de comandos
} while (expr);
```



```
/* Fatorial (versao 3) */
#include <stdio.h>
int main (void)
{
    int k;
    int n;
    int f = 1;
    /* requisita valor do usuário até um número não negativo ser informado */
    do
    { printf("Digite um valor inteiro nao negativo:");
        scanf ("%d", &n);
    } while (n<0);
    /* calcula fatorial */
    for (k = 1; k <= n; k++)
        f *= k;
    printf(" Fatorial = %d\n", f);
    return 0;
}
```



```
/* Fatorial (versao 4) */
#include <stdio.h>
int main (void)
{
    int k;
    int n;
    int f = 1;
    /* O que faz este programa? */
    do {
        printf("Digite um valor inteiro nao negativo:");
        scanf ("%d", &n);
        /* calcula fatorial */
        for (k = 1; k <= n; k++)
            f *= k;
        printf(" Fatorial = %d\n", f);
    } while (n>=0);
    return 0;
}
```



Bash no Ubuntu no Windows

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define QL printf("\n")
5
6 int main(){
7     int num, i;
8     char c;
9
10    do{
11        printf("Digite um número inteiro: ");
12        scanf(" %d", &num);
13
14        printf("Tabuada de %d\n", num);
15        for(i=1;i<=10;i++){
16            printf("%d x %d = %d\n", num, i, num * i);
17        }
18        QL;
19        printf("Continua: (S/N)?");
20        scanf(" %c", &c);
21        QL;
22    }while((c=='s')||(c=='S'));
23    QL;
24
25    getchar();
26 }
```



# Laços

- Interrupção de laços – instrução “break”
- Termina a execução do laço

```
#include <stdio.h>
int main (void)
{
    int i;
    for (i = 0; i < 10; i++) {
        if (i == 5)
            break;
        printf("%d ", i);
    }
    printf("fim\n");
    return 0;
}
```

A saída deste programa, se executado, será: 0 1 2 3 4 fim



# Use break para fugir...

Você pode criar loops que verificam uma condição no início ou fim do código do loop. Mas e se quiser fugir do loop em algum ponto no meio? Você pode sempre reestruturar seu código, mas às vezes é mais simples fugir do loop imediatamente usando a declaração **break**:

```
while(feeling_hungry) {  
    eat_cake();  
    if (feeling_queasy) {  
        /* Break out of the while loop */  
        break;  
    }  
    drink_coffee();  
}
```

“break” sai  
imediatamente  
do loop.

Uma declaração **break** vai te tirar diretamente do loop atual, pulando seja o que for que tiver a seguir no código de loop. **breaks** podem ser úteis, porque às vezes são a melhor forma de terminar um loop. Mas, talvez, queira evitar usar muitos, porque também deixam o código um pouco mais difícil de ler.



Veja bem!

A declaração  
**break** é usada  
para sair de loops  
e declarações  
**switches**.

Certifique-se de que saiba do que  
está saindo quando usar o **break**.

...e continue continuando

Se quiser pular o resto do loop e voltar para o início deste loop, a declaração **continue** é sua amiga:

```
while(feeling_hungry) {  
    if (not_lunch_yet) {  
        /* Go back to the loop condition */  
        continue;      “continue” te leva para  
    }                o início do loop.  
    eat_cake();  
}
```

# Interrupção de laços

- Comando “continue”
- Termina a iteração corrente e passa para a próxima

```
#include <stdio.h>

int main (void)
{
    int i;
    for (i = 0; i < 10; i++ ) {
        if (i == 5) continue;
        printf("%d ", i);
    }
    printf("fim\n");
    return 0;
}

gera a saída: 0 1 2 3 4 X 6 7 8 9 fim
```



# Interrupção de laços

- Deve se ter cuidado para não criar uma iteração eterna – loop infinito

```
/* INCORRETO */
#include <stdio.h>
int main (void)
{
    int i = 0;
    while (i < 10) {
        if (i == 5) continue;
        printf("%d ", i);
        i++;
    }
    printf("fim\n");
    return 0;
}
```

cria “iteração eterna” pois i não será mais incrementado quando chegar a 5





## Contos da Cripta

**breaks não fogem de  
declarações if.**

*No dia 15 de janeiro de 1990, o sistema de telefonia de longa distância da AT&T's falhou, e 60 mil pessoas perderam o serviço de telefonia. A causa? Um desenvolvedor trabalhando no código C, usado nas transferências, tentou usar um break para sair de uma declaração if. Mas breaks não saem de ifs. Ao invés disso, o programa pulou uma seção inteira de código e criou um bug que interrompeu 70 milhões de ligações por mais de nove horas.*

# Comando “switch”

- Seleciona um entre vários “casos”, opções. (“opn” deve ser um **inteiro** ou **caractere**)
- Não esquecer de colocar “**break**” no final de cada bloco de comandos. (Exceto **default** – não é necessário, mas pode usar.)

```
switch ( expr )
{
    case op1: bloco de comandos 1; break;
    case op2: bloco de comandos 2; break;
    ...
    default:   bloco de comandos default; break;
}
```



# Calculadora 5 operações

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #define QL printf("\n")
5 main(){
6     float num1, num2;
7     char c, op;
8     do{
9         printf("\nDigite: numero OP numero\n");
10        scanf("%f %c %f", &num1, &op, &num2);
11        switch(op)
12        {
13            case '+':{
14                printf("%g\n", num1+num2);
15                printf("\nA operacao foi de adicao\n");
16            }break;
17            case '-': printf("%g\n", num1-num2); break;
18            case '*': printf("%g\n", num1*num2); break;
19            case '/': printf("%g\n", num1/num2); break;
20            case '^': printf("%g\n", pow(num1,num2)); break;
21            default:   printf("Operador Invalido\n"); break;
22        }
23        QL;
24        printf("Continua(S/N)?");
25        scanf(" %c", &c);
26    }while((c == 'S')||(c == 's'));
27    QL;
28    system("pause");
29 }
```



*Veja bem!*

.....

## **A ausência de breaks podem deixar o seu código cheio de bugs.**

*A maioria dos programas em C tem um break,  
no fim de cada seção case, para deixar o código  
mais fácil de entender, mesmo a custo de  
menos eficiência.*

.....



## PONTOS DE BALA

- Declarações `switch` podem substituir uma sequência de declarações `if`.
- Declarações `switch` verificam um único valor.
- O computador começa a executar o código a partir da primeira declaração `case` correspondente.
- Ele continua a executar o código até chegar em um `break` ou no final da declaração `switch`.
- Certifique-se de que tenha incluído `breaks` nos lugares certos; caso contrário, seus `switches` podem ficar cheios de bugs.

## *não existem* Perguntas Idiotas

**P:** Por que usar uma declaração `switch` ao invés de uma `if`?

**R:** Se estiver realizando múltiplas verificações na mesma variável, você talvez queira usar a declaração `switch`.

**P:** Quais são as vantagens de usar uma declaração `switch`?

**R:** Há várias. Primeira: clareza. Fica claro que um bloco inteiro de código está processando uma única variável. Isso não fica muito óbvio se estiver usando uma sequência de declarações `if`. Segunda: você

pode usar a lógica em sequência e reutilizar segmentos de código para casos diferentes.

**P:** A declaração `switch` precisa verificar uma variável? Não pode verificar um valor?

**R:** Sim, ela pode. A declaração `switch` simplesmente verifica se dois valores são iguais.

**P:** Posso verificar strings em uma declaração `switch`?

**R:** Não, você não pode usar uma declaração `switch` para verificar uma string de caracteres ou qualquer tipo de array. A declaração `switch` apenas verifica um único valor.

# Resumo – Controle de fluxo

```
if ( expr ) { bloco de comandos } else { bloco de comandos }
```

```
condição ? expressão1 : expressão2;
```

```
while ( expr ) { bloco de comandos }
```

```
for ( expr_inicial; expr_boleana; expr_de_incremento ) { bloco de comandos }
```

```
do { bloco de comandos } while ( expr );
```

```
switch ( expr ) {
    case op1: bloco de comandos 1; break;
    case op2: bloco de comandos 2; break;
    ...
    default:   bloco de comandos default; break;
}
```



# Observação: Para acentuar

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include <locale.h> //biblioteca de localização
5
6 main()
7 {
8     //Instrução para localizar o programa
9     setlocale(LC_ALL, "Portuguese");
10
11    //Teste da acentuação
12    printf("Alô mundo Ç ã! \n\n");
13
14    system("pause");
15
16 }
```



**VER A LISTA DE  
EXERCÍCIOS QUE ESTARÁ  
DISPONÍVEL NO BLOG E NO  
DROPBOX.**

