

ICMC USP

1o.semestre/2010

Introdução à Ciência de Computação

SCC-120

Aula 3: Comandos em C

Profa. Roseli Ap. Francelin Romero

O comando *if*

if (*expressão é verdadeira*)

 execute comando ou bloco de comandos ;

else /* se expressão é falsa */

 execute comando ou bloco de comandos ;

Ex:

if (count > 9)

 count = 0;

else

 count++;

Aninhamento de *if*

É possível aninhar construções do tipo if-else em diversos níveis

```
if (cond1)                /* if1 */
|
|  if (cond2)              /* if2 */
|  |  comando if2;
|  |
|  |  else                /* else2 */
|  |  |  comando else2;
|  |
|  |  else                /* else1 */
|  |  |  if (cond3)       /* if3 */
|  |  |  |  if (cond4)   /* if4 */
|  |  |  |  |  comando if4;
|  |  |  |  |
|  |  |  |  |  else      /* else4 */
|  |  |  |  |  |  comando else4;
|  |  |  |
|  |  |  else            /* else3 */
|  |  |  |  comando else3;
|  |
|  else
|  |  comando else1;
```

if - exemplo

```
#include <stdio.h>
void main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num > 10)
        printf ("\n\n O numero e maior que 10");
    if (num == 10)
    {
        printf ("\n\n Voce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    if (num < 10)
        printf ("\n\n O numero e menor que 10");
}
```

O comando *if*

Podemos pensar no comando **else** como sendo um complemento do comando **if**. O comando **if** completo tem a seguinte forma geral:

```
if (condição) {  
    seqüência_de_comandos_1;  
}  
else {  
    seqüência_de_comandos_2;  
}
```

Aninhamento de *if*

O **if** aninhado é simplesmente um **if** dentro da declaração de um outro **if** externo. O único cuidado que devemos ter é o de saber exatamente a qual **if** um determinado **else** está ligado.

Aninhamento de *if*

```
#include <stdio.h>
void main ()
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d", &num);
    if (num == 10)
    {
        printf ("\n\n Voce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        if (num > 10)
            printf ("O numero e maior que 10.");
        else
            printf ("O numero e menor que 10.");
    }
}
```

Aninhamento de *if*

Observe sempre a correspondência entre *if*'s e *else*'s

```
if (cond1)
→ if (cond2)
    comando if2;
else
    comando if1;          /* atenção: else2! */
                          /* erro: comando do if2 */
```

modo correto:

```
if (cond1) {
    if (cond2)
        comando if2;
}
else
    comando if1;
```


O *else*

A expressão da condição será avaliada:

- ◆ Se ela for diferente de zero, a **seqüência_comandos_1** será executada.
- ◆ Se for zero a **seqüência_comandos_2** será executada.

É importante nunca esquecer que, quando usamos a estrutura **if-else**, estamos garantindo que uma das duas declarações será executada.

if - exemplo

```
#include <stdio.h>
void main ( )
{
    int num;
    printf ("Digite um numero: ");
    scanf ("%d", &num);
    if (num == 10)
    {
        printf ("\n\n Voce acertou!\n");
        printf ("O numero e igual a 10.\n");
    }
    else
    {
        printf ("\n\n Voce errou!\n");
        printf ("O numero e diferente de 10.\n");
    }
}
```

Outro exemplo

- Converter um *string* tipo “10” para um valor binário

```
...  
char str[] = "10";  
if (str[0]=='\0')  
    if (str[1]=='\0') printf("Zero");  
    else printf("Um");  
else      /* str[0] == '1' */  
    if (str[1]=='\0') printf("Dois");  
    else printf("Tres");  
...
```

Encadeamento *if-else-if*

```
if (teste_1) <comando_1>;  
else if (teste _2) <comando _2>;  
else if (teste _3) <comando _3>;  
...  
else <comando _n>;
```

- No encadeamento apenas *um* dos n comandos será executado: o primeiro cujo teste for verdadeiro

Encadeamento *if-else-if*

A estrutura **if-else-if** é apenas uma extensão da estrutura **if-else**. Sua forma geral é:

```
if (condição_1) {  
    seqüência_de_comandos_1;  
}  
else if (condição_2) {  
    seqüência_de_comandos_2;  
}  
...  
else if (condição_n) {  
    seqüência_de_comandos_n;  
}  
else {  
    seqüência_de_comandos_default;  
}
```

else-if - exemplo

```
#include <stdio.h>
void main ()
{
    int num;

    printf ("Digite um numero: ");
    scanf ("%d",&num);
    if (num > 10)
        printf ("\n\n O numero e maior que 10");
    else if (num == 10)
    {
        printf ("\n\n Voce acertou!\n");
        printf ("O numero e igual a 10.");
    }
    else if (num < 10)
        printf ("\n\n O numero e menor que 10");
}
```

Encadeamento *if-else-if*

Exemplo: escrever o nome de um dígito

'0' -> "zero", '1' -> "um", etc.

```
...  
if (ch == '0') printf("Zero");  
else if (ch=='1') printf("Um");  
else if (ch=='2') printf("Dois");  
else if ...  
else if (ch=='9') printf("Nove");  
else printf("Nao era um digito!");  
...
```

A Expressão Condicional

Quando o compilador avalia uma condição, ele quer um valor de retorno para poder tomar a decisão. Mas esta expressão não necessita ser uma expressão no sentido convencional. Uma variável sozinha pode ser uma "expressão" e esta retorna o seu próprio valor. Assim:

```
int num;  
if (num!=0) ....  
if (num==0) ....  
for (i = 0; string[i] == '\0'; i++)
```

equivalem a

```
int num;  
if (num) ....  
if (!num) ....  
for (i = 0; string[i]; i++)
```


O Operador ?

Uma expressão como:

```
if (a > 0)
    b = -150;
else
    b = 150;
```

pode ser simplificada usando-se o operador **?** da seguinte maneira:

```
b = a > 0 ? -150 : 150;
```

Expressão Condicional ?

- A expressão condicional “? :” é uma simplificação do if-else utilizada tipicamente para atribuições condicionais:

`exp1?exp2:exp3` \approx `if (exp1)?exp2; else exp3;`

- Ex:implementando $z = \max(x, y)$ com:

if: `if (x > y) z=x; else z=y;`

?: `z = (x > y) ? x : y;`

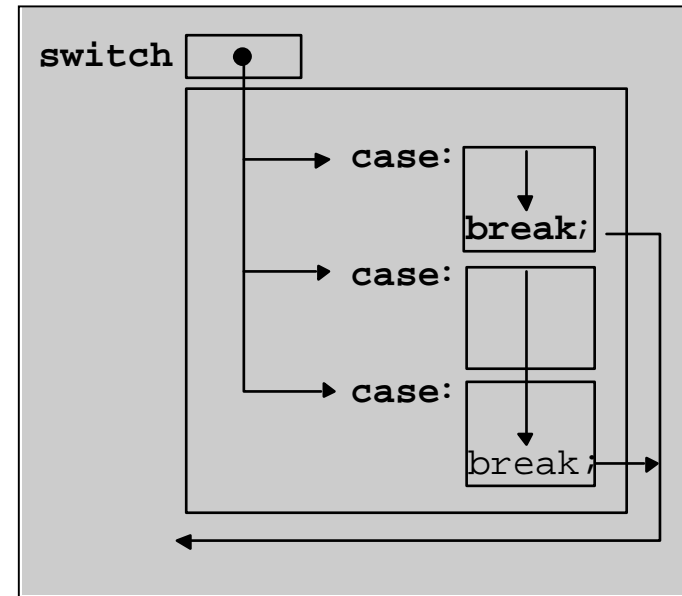
Exemplo

- O que faz o trecho de código abaixo ?

```
...  
conta = 0;  
for (index=0; index < 1000; index++)  
{  
    printf("%d", conta);  
    conta = (conta==8) ? 0 : conta+1;  
}
```

O comando *switch*

```
switch ( valor ) {  
    case valor1:  
        comandos1;  
        break;  
    case valork:  
        comandosk;  
        break;  
    default:  
        comandos_default;  
        break;  
}
```



O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos.

O comando *switch*

- a expressão *valor* é avaliada e o valor obtido é comparado com os valores associados às cláusulas *case* em sequência.
- quando o valor associado a uma cláusula é igual ao valor do *switch* os respectivos comandos são executados até encontrar um *break*.
- se não existir um *break* na cláusula selecionada, os comandos das cláusulas seguintes são executados em ordem até encontrar um *break* ou esgotarem-se as cláusulas do *switch*
- se nenhuma das cláusulas contém o valor de seleção, a cláusula *default*, se existir, é executada

Exemplo *switch*

```
switch( char_in ) {  
    case '.': printf( "Ponto.\n" );  
                break;  
    case ',': printf( "Virgula.\n" );  
                break;  
    case ':': printf( "Dois pontos.\n" );  
                break;  
    case ';': printf( "Ponto e virgula.\n" );  
                break;  
    default : printf( "Nao eh pontuacao.\n" );  
}
```

Exemplo *switch*

```
switch( char_in ) {  
    case '0': putchar('0');      /* 0123456789 */  
    case '1': putchar('1');      /* 123456789 */  
    case '2': putchar('2');      /* 23456789 */  
    case '3': putchar('3');      /* 3456789 */  
    case '4': putchar('4');      /* 456789 */  
    case '5': putchar('5');      /* 56789 */  
    case '6': putchar('6');      /* 6789 */  
    case '7': putchar('7');      /* 789 */  
    case '8': putchar('8');      /* 89 */  
    case '9': putchar('9');      /* 9 */  
        break; }
```

Exercício *switch*

- Contar o número de ocorrências de dígitos decimais em uma sequência de caracteres digitados pelo usuário utilizando o comando switch

```
main() {  
    char ch; int ch_count = 0;  
    printf("- Entre caracteres ('F' para terminar) -\n" );  
    do {    ch = getchar();  
        /*  
            usar switch para contar os digitos  
        */  
    } while (ch != 'F');  
    printf ("\nLidos: %d\n", ch_count);  
}
```

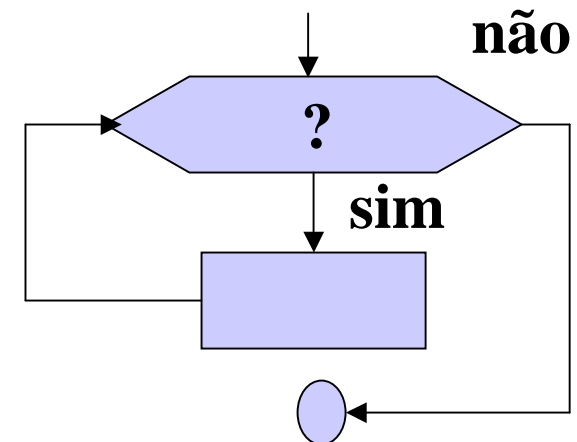

Repetição por Condição

- Uma das formas de repetir um conjunto de comandos de um algoritmo é subordiná-lo a um comando de repetição usando uma estrutura da forma:

enquanto *condição* faça

comandos

fim enquanto



Repetição por Condição

- Os comandos serão repetidos zero ou mais vezes, enquanto o valor da condição for verdadeiro.
- Essa estrutura normalmente é denominada *laço* ou *loop*.

Repetição por Condição – Funcionamento

- A condição da cláusula *enquanto* é testada.
- Se ela for verdadeira os comandos seguintes são executados em sequência como em qualquer algoritmo, até a cláusula *fim enquanto*.
- O fluxo nesse ponto é desviado de volta para a cláusula *enquanto*.
- Se a condição agora for falsa (ou quando finalmente for), o fluxo do algoritmo é desviado para o primeiro comando após a cláusula *fim enquanto*.
- Se a condição ainda for verdadeira, o processo se repete.

Repetição por Condição

- A condição pode ser qualquer expressão que resulte em um valor do tipo lógico e pode envolver operadores aritméticos, lógicos, relacionais e resultados de funções.

Exemplo

Algoritmo calcula_senos

variável

n, i: inteiro

leia(n)

$i \leftarrow 0$

enquanto $i \leq n$ faça

 escreva(i, seno(i))

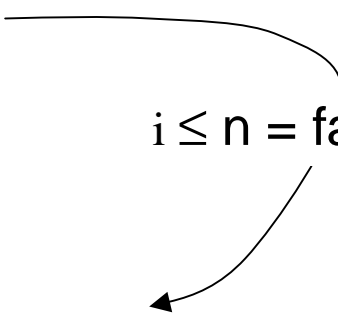
$i \leftarrow i + 1$

fim enquanto

 escreva ('fim do cálculo')

fim

$i \leq n = \text{falso}$



Exemplo

Algoritmo calcula_senos

variável

n, i: inteiro

leia(n)

$i \leftarrow 0$

enquanto $(i < 45)$ e $(i < n + 1)$ faça

 escreva(i, seno(i))

$i \leftarrow i + 1$

fim enquanto

 escreva ('fim do cálculo')

fim

Loop Infinito

enquanto *verdadeiro* faça

comandos

fim enquanto

- Um loop infinito pode acontecer também quando cometemos algum erro ao especificar a condição lógica que controla a repetição ou ao esquecer de algum comando dentro da iteração.

Exemplo

Algoritmo calcula_senos

variável

n, i: inteiro

leia(n)

$i \leftarrow 0$

enquanto $(i < 45)$ e $(i < n + 1)$ faça

 escreva(seno(i))

fim enquanto

fim

A variável i não é incrementada

Exemplo – soma consecutiva de dados sem estrutura de repetição

Algoritmo somasimples

variável

valor1, valor2, valor2, valor4: real

soma: real

leia(valor1, valor2, valor3, valor4)

soma \leftarrow (valor1 + valor2 + valor3 + valor4)/4

escreva(soma)

fim

Exemplo – soma consecutiva de dados

Algoritmo somasimples

variável

valor, soma: real

soma \leftarrow 0

leia(número)

enquanto número $>$ -100 faça

 soma \leftarrow soma + número

 leia(número)

fim enquanto

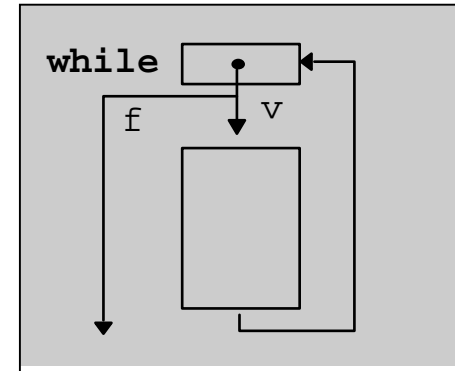
escreva(soma)

fim

Exercício: Além da soma, calcular a média aritmética

Comando *while*

```
while (condição) {  
    comandos;  
}
```



- 1º avalia condição
- se condição é verdadeira, executa comandos do bloco
- ao término do bloco, volta a avaliar condição
- repete o processo até que condição seja falsa

Comando *while*

O comando **while** que tem a seguinte forma geral:

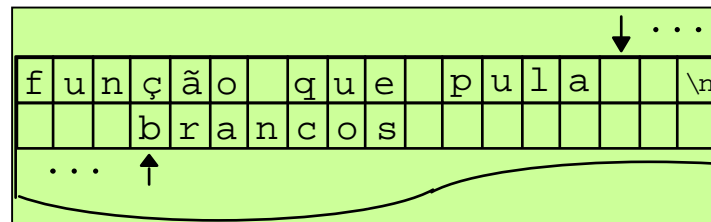
```
while (condição) {  
    seqüência_de_comandos;  
}
```

seria equivalente a:

```
if (condição)  
{  
    seqüência_de_comandos;  
    "Volte para o comando if"  
}
```

Exemplo *while*

```
void pula_branços () {  
    int ch;  
    while ((ch = getchar()) == ' ' || /* brancos */  
           ch == '\n' ||             /* newline */  
           ch == '\t' )              /* tabs */  
        ; /* não faz nada */  
}
```

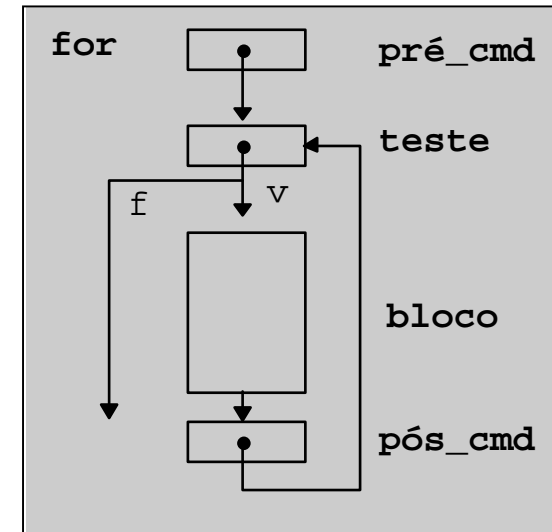


Comando *for*

```
for (pré_cmd; teste; pós_cmd) {  
    comandos;  
}
```

- em termos de while, equivale a:

```
pré_cmd;  
while (teste) {  
    comandos;  
    pós_cmd;  
}
```



Comando *for*

O loop `for` é usado para repetir um comando, ou bloco de comandos, diversas vezes, de maneira que se possa ter um bom controle sobre o loop.

```
for (inicialização; condição; incremento) {  
    seqüência_de_comandos;  
}
```

Comando *for*

- 1º executa pré_cmd (**inicialização**), que permite iniciar variáveis
- 2º avalia teste (**condição**): se verdadeiro, executa comandos do bloco, senão encerra laço
- ao término do bloco, executa pós_cmd (**incremento**)
- reavalia teste
- repete o processo até que teste seja falso

Exemplo *for*

Imprimir o conteúdo de um vetor:

```
void main( ) {  
  int i;  
    for ( i=0; i < size_array; i++)  
      printf ("%d ", array[i]);  
}
```

Comando *for*

- Podemos omitir qualquer um dos elementos (*inicialização*, *condição* ou *incremento*) do **for**.

Ex.: *for* (*inicialização*; ;*incremento*) {
 seqüência de comandos;
}

- Este é um loop infinito porque será executado para sempre (não existindo a condição, ela será sempre considerada verdadeira), a não ser que ele seja interrompido.
- Para interromper um loop como este usamos o comando **break**.

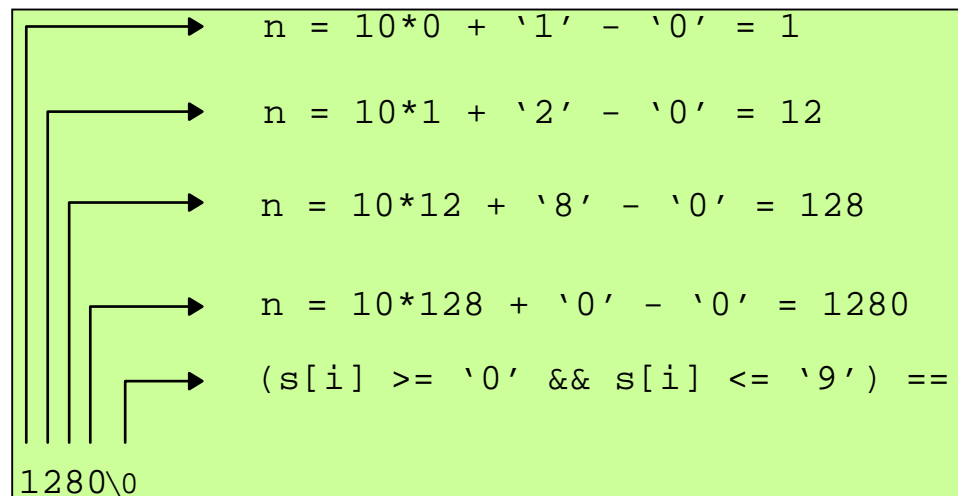
Exemplo *for*

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int count;
    char ch;
    for (count = 1; ; count++)
    {
        ch = getch();
        if (ch == 'X')
            break;
        printf("\n Letra: %c", ch);
    }
}
```

Exemplo *for*

Conversão de string para inteiro

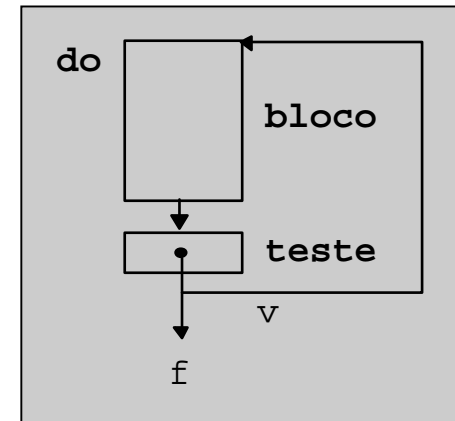
```
int atoi (char s[]) {      /* asc to integer */  
    int i, n;  
    for (n=0, i=0; s[i] >= '0' && s[i] <= '9'; ++i)  
        n = 10 * n + s[i] - '0';  
    return (n);  
}
```



Comando *do-while*

- do-while é utilizado sempre que o bloco de comandos deve ser executado ao menos uma vez

```
do {  
    comandos;  
} while (condição);
```



Comando *do-while*

- 1º executa comandos
- 2º avalia condição:
 - se verdadeiro, reexecuta comandos do bloco
 - senão encerra laço

Exemplo

do-while

```
#include <stdio.h>
void main ( )
{
    int i;
    do {
        printf ("\n Escolha a fruta pelo numero:\n");
        printf ("\t (1)...Mamao\n");
        printf ("\t (2)...Abacaxi\n");
        printf ("\t (3)...Laranja\n\n");
        scanf("%d", &i);
    } while ((i < 1) || (i > 3));

    switch (i) {
        case 1:
            printf ("\t\t Voce escolheu Mamao.\n");
            break;
        case 2:
            printf ("\t\t Voce escolheu Abacaxi.\n");
            break;
        case 3:
            printf ("\t\t Voce escolheu Laranja.\n");
            break;
    }
}
```

Exemplo *do-while*

```
void itoa (int num, char s[]) {  
  int i = 0; int sinal;  
  if ((sinal = num) < 0) /* armazena sinal */  
    num = - num; /* faz num positivo */  
  do {  
    s[i] = num % 10 + '0'; /* unidades */  
    ++i;  
  } while ((num /= 10) > 0); /* quociente */  
  if (sinal < 0) {  
    s[i] = '-';  
    ++i;  
  }  
  s[i] = '\\0';  
}
```


Comando *break*

- o comando *break* permite interromper a execução de um laço ou de um *switch*
- Ex:

```
main () {  
    int i, j;  
    for (i = 0; i < 4; i++)  
        for (j = 0; j < 2; j++)  
            if (i == 1) break;  
            else printf("i: %d j: %d\n", i, j);  
}
```

i: 0 j: 0
i: 0 j: 1
i: 2 j: 0
i: 2 j: 1
i: 3 j: 0
i: 3 j: 1

Comando *continue*

- o comando *continue* leva a execução do próximo passo de uma iteração. Os comandos que sucedem *continue* no bloco não são executados

- Ex:

```
main() {  
  int i;  
  for (i = 0; i < 5; i++)  
    if (i == 1) continue;  
    else printf("i: %d \n", i);  
}
```

i: 0
i: 2
i: 3
i: 4

Comando *continue*

- O comando **continue** pode ser visto como sendo o oposto do **break**;
- Ele só funciona dentro de um loop.
- Quando o comando **continue** é encontrado, o loop pula para a próxima iteração, sem o abandono do loop, ao contrário do que acontecia no comando **break**.

Comando *continue*

```
#include <stdio.h>
void main()
{
    int opcao;
    while (opcao != 4) {
        printf("\n\n Escolha uma opcao entre 1 e 4: ");
        scanf("%d", &opcao);
        if ((opcao > 4) || (opcao < 1))
            continue;
        /* Opcao invalida: volta ao inicio do loop */
        switch (opcao) {
            case 1: printf("\n --> Primeira opcao..");
                    break;
            case 2: printf("\n --> Segunda opcao..");
                    break;
            case 3: printf("\n --> Terceira opcao..");
                    break;
            case 4: printf("\n --> Abandonando..");
                    break;
        } /* fim -switch */
    } /* fim-while */
} /* fim-main */
```

Goto's e labels

- C suporta os comandos *goto*, que permitem o desvio do fluxo de execução do programa para uma posição indicada por um rótulo (*label*)
- apesar de banido da prática de programação estrutura, *goto's* podem ser úteis em determinadas circunstâncias, como sair de dentro de laços aninhados

Goto

```
#include <stdio.h>
void main()
{
    int opcao;
    while (opcao != 4)
    {
        REFAZ: printf("\n\n Escolha uma opcao entre 1 e 4: ");
        scanf("%d", &opcao);
        if ((opcao > 4)|| (opcao <1)) goto REFAZ;
        /* Opcao invalida: volta ao rotulo REFAZ */
        switch (opcao)
        {
            ...
        }
    }
}
```

Exemplo *goto*

```
for ( ... )  
for ( ... ) {  
    ...  
    if ( desastre )  
        goto erro;  
}  
  
...  
/* o label deve estar na mesma função */  
erro:  
    dah_um_jeitinho();
```

Exemplo com *goto*

```
/* usando goto */  
for ( i=0; i < n; i++ )  
    for ( j=0; j < m; j++ )  
        if ( A[i] == B[j] )  
            goto achei;  
/* trata outro caso: não achou */  
achei:  
/* tratamento do achado */
```


Exemplo sem *goto*

```
/* sem goto */  
int achei = 0;  
for ( i=0; i < n; i++ )  
    for ( j=0; j < m; j++ )  
        if ( A[i] == B[j] )  
            achei = 1;  
if (achei) /* tratamento do achado */  
else /* trata outro caso: não achou */
```

Exercícios

1. Usando o comando *for*, faça um algoritmo que conte o número de 1's que aparecem em um *string*
ex: 0011001 => 3
2. Usando o comando *while*, escreva um programa que substitui as ocorrências de um caracter *ch0* em um string por um outro caracter *ch1*
3. Utilizando o comando *do-while*, implemente um programa que converte um *string* contendo um número binário positivo em um inteiro.
ex: "001101" => 13

Slides cedidos pela profa. Renata Fortes e revisados e/ou modificados
pela Profa. Roseli Romero
SCE-ICMC-USP