O 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. Sua forma geral é:

```
for (inicialização; condição; incremento) {
        seqüência de comandos;
}
```

O Comando for

- Abaixo vemos um programa que coloca os primeiros 100 números inteiros na tela:

```
#include <stdio.h>
void main ( )
{
        int count;
        for (count = 1; count <= 100; count++)
                printf ("%d ", count);
}
```

O Comando for

- O melhor modo de se entender o loop for é ver como ele funciona "por dentro". O loop for é equivalente a se fazer o seguinte:

```
inicialização;
 if (condição) {
     seqüência de comandos;
     incremento;
     "Volte para o comando if"
 }
```

1) Executa a inicialização incondicionalmente e testa a condição.
2) Se a condição for falsa ele não faz mais nada. Se a condição for verdadeira ele executa a seqüência de comandos;
3) Faz o incremento e volta a testar a condição.
4) Ele fica repetindo estas operações até que a condição seja falsa.

O 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

```
#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);
        }
}
```

■ O loop sem conteúdo

for (inicialização;condição;incremento);

Uma das aplicações desta estrutura é gerar tempos de espera.

```
#include <stdio.h>
void main ()
{
    long int i;
    printf("\a");              /* Imprime o caracter de alerta (um beep) */
    for (i=0; i < 10000000; i++);  /* Espera 10.000.000 de iterações */
    printf("\a");              /* Imprime outro caracter de alerta */
}
```

O Comando while

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

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

- O **while** seria equivalente a:

```
if (condição)
{
    seqüência de comandos;
    "Volte para o comando if"
}
```

O Comando while

Exemplo

```
#include <stdio.h>
void main ()
{
        char Ch;

        Ch='\0';
        while (Ch != 'q')
        {
                Ch = getch();
        }
}
```

O Comando do-while

- O **do-while** tem a forma geral:

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

eqüivale a

```
seqüência de comandos;
if (condição)
    "Volta para a seqüência de comandos"
```

- **do-while** garante que o corpo dos comandos será executada pelo menos uma vez.

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

    do
    {
        printf ("\n\n Escolha a fruta pelo numero:\n\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));
```

cont.

```
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;
}
}
```

O Comando break

- Nós já vimos dois usos para o comando **break**: interrompendo os comandos **switch** e **for**.
- Na verdade, o **break** pode quebrar a execução de um comando (como no caso do **switch**) ou interromper a execução de qualquer loop (**for**, **while** ou **do-while**).
- O **break** faz com que a execução do programa continue na primeira linha seguinte ao loop ou bloco que está sendo interrompido.

O 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**.

```
#include <stdio.h>
void main()
{
      int opcao;
      while (opcao != 5)
      {
            printf("\n\n Escolha uma opcao entre 1 e 5: ");
            scanf("%d", &opcao);
            if ((opcao > 5) || (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 --> Quarta opcao..");
                        break;
                    case 5:
                        printf("\n --> Abandonando..");
                        break;
                }
            }
        }
```

O Comando goto

- O **goto** realiza um salto incondicional para um local especificado. Este local é determinado por um rótulo.
- Um rótulo, na linguagem C, é uma marca no programa. Você dá o nome que quiser a esta marca.
- Podemos escrever uma forma geral:

```
nome_do_rótulo:
    ....

    goto nome_do_rótulo;
    ....
```

O Comando goto

- O **goto** pode saltar para um rótulo que esteja mais à frente ou para trás no programa;
- Uma observação importante é que o rótulo e o **goto** devem estar dentro da mesma função.

O Comando goto

- O seguinte exemplo do uso do **goto** é equivalente ao comando **for**:

```
    inicialização;
início_do_loop:
    if (condição)
    {
      sequencia de comandos;
      incremento;
      goto início_do_loop;
    }
```

O Comando goto

Algumas Observações

- O comando goto deve ser utilizado com parcimônia, pois o abuso no seu uso tende a tornar o código confuso.
- O goto não é um comando necessário, podendo sempre ser substituído por outras estruturas de controle.
- Puristas da programação estruturada recomendam que o goto nunca seja usado.

O Comando goto

- Em algumas situações muito específicas o comando **goto** pode tornar um código mais fácil de se entender se ele for bem empregado.
- Um caso em que ele pode ser útil é quando temos vários loops e **if**s aninhados e se queira, por algum motivo, sair destes loops e **if**s todos de uma vez. Neste caso um **goto** resolve o problema mais elegantemente que vários **break**s, sem contar que os **break**s exigiriam muito mais testes.

O Comando goto

Exemplo

```
#include <stdio.h>
void main()
{
    int opcao;
    while (opcao != 5)
    {
        REFAZ: printf("\n\n Escolha uma opcao entre 1 e 5: ");
        scanf("%d", &opcao);
        if ((opcao > 5)||(opcao <1)) goto REFAZ;
                    /* Opcao invalida: volta ao rotulo REFAZ */
        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 --> Quarta opcao..");
                    break;
                case 5:
                    printf("\n --> Abandonando..");
                    break;
            }
        }
    }
```