## ESTRUTURAS DE CONTROLE DE FLUXO

**6 ESTRUTURAS DE REPETIÇÃO**

**For** é a primeira de uma série de três estruturas para se trabalhar com loops de repetição. As outras são  [**while**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c440.html) e  [**do**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c450.html). As três compõem a segunda família de comandos de controle de fluxo. Podemos pensar nesta família como sendo a das estruturas de repetição controlada.

### - Estrutura For

O loop (laço) 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) <ação>;*

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)*

*{*

*<ação>;   
<incremento>;   
<"Volte para o comando if">   
}*

Podemos ver, então, que o **for** executa a <inicialização> incondicionalmente e testa a <condição>. Se a <condição> for falsa ele não faz mais nada. Se a <condição> for verdadeira ele executa a <ação>, faz o <incremento> e volta a testar a <condição>. Ele fica repetindo estas operações até que a <condição> seja falsa.

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

#include <stdio.h>

#include <stdlib.h>

int main ()

{

int cont;

for (cont=1; cont<=100; cont++)

printf ("%d ",cont);

system("PAUSE");

return(0);

}

Note que, no exemplo acima o incremento da variável **cont** é feito usando o operador de incremento que nós agora já conhecemos. Esta é a forma usual de se fazer o incremento (ou decremento) em um loop **for**.

Podemos também decrementar o contador e teremos os números em ordem decrescente:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int cont;

for (cont=100; cont>=0; cont=cont-1)

printf ("%d ",cont);

system("PAUSE");

return 0;

}

Como outro exemplo temos esse programa que imprime o alfabeto em letras maiúsculas

#include <stdio.h>

#include <stdlib.h>

int main()

{

char letra;

for(letra = 'A' ; letra <= 'Z' ; letra =letra+1)

printf("%c ", letra);

system("PAUSE");

return 0;

}

Este programa funciona porque as letras maiúsculas de A a Z possuem código inteiro sequencial.

O **for** na linguagem C é  bastante flexível. Temos acesso à inicialização, à condição e ao incremento. Qualquer uma destas partes do **for** pode ser uma expressão qualquer do C, desde que ela seja válida. Isto nos permite fazer o que quisermos com o comando. As três formas do for abaixo são válidas:

for ( count = 1; count < 100 ; count++) { ... }

for (count = 1; count < NUMERO\_DE\_ELEMENTOS ; count++) { ... }

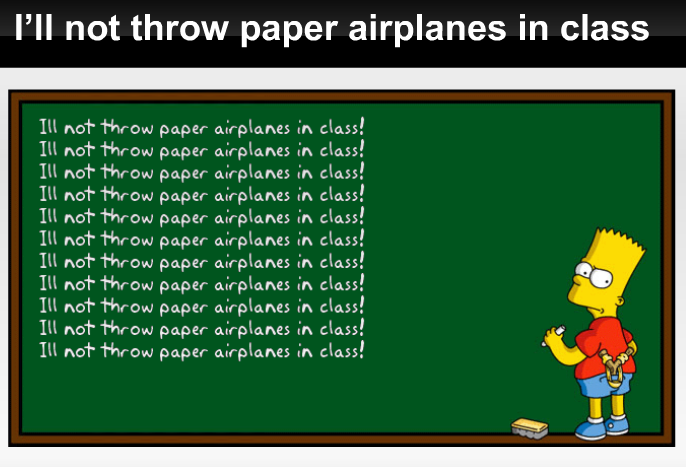
for (count = 1; count < BusqueNumeroDeElementos() ; count+=2) { ... }

etc ...

Preste atenção ao último exemplo: o incremento está sendo feito de dois em dois. Além disto, no teste está sendo utilizada uma função (BusqueNumeroDeElementos() ) que retorna um valor que está sendo comparado com count.

EXERCÍCIOS

* 1. Ajude o Bart Simpson a escrever 500 vezes a frase do castigo, usando o laço for.



* 1. Escrever um programa que solicite um valor entre 1 e 10 e mostre sua respectiva tabuada.
  2. Elabore um algoritmo que mostre todos os números inteiros de 100 a 200 inclusive, e ao final mostra o seu somatório
  3. Elabore um algoritmo que leia 20 números inteiros. Ao final, devem ser informados quantos valores maiores que 10 foram lidos e a sua média.

### 

### O Comando while

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

*while (condição) <ação>;*

Assim como fizemos para o comando **for**, vamos tentar mostrar como o **while** funciona fazendo uma analogia. Então o **while** seria equivalente a:

*if (condição)*

*{*

*<ação>;*   
*<"Volte para o comando if">*   
*}*

Podemos ver que a estrutura **while** testa uma <condição>. Se ela for verdadeira a <ação> é executada e faz-se o teste novamente, e assim por diante.

Vamos ver um exemplo do uso do **while**.   O programa abaixo é executado enquanto i for menor que 100.

#include <stdio.h>

#include <stdlib.h>

int main ()

{

int i = 0;

while ( i < 100)

{

       printf(" %d", i);

i++;

       }

system(“PAUSE”);

return(0);

}

O programa abaixo espera o usuário digitar a pressionar 'q' e só depois finaliza:

#include <stdio.h>

#include <stdlib.h>

int main ()

{

char Ch;

Ch='\0';

while (Ch != 'q')

{

       scanf("%c", &Ch);

       }

system(“PAUSE”);

return(0);}

EXERCÍCIOS

* 1. (WHILE, GETCHAR/GETCH/GETCHE(para 1 char)) Construir um algoritmo que solicite um número inteiro. Em seguida o algoritmo deve multiplicar por 2 número lido, mostrar o resultado e continuar multiplicando e mostrando os resultados enquanto o usuário responder Sim (S) a pergunta “Deseja continuar (S/N) ?” (Use a função *toupper* para converter caracteres em maiúsculos ).
  2. (WHILE, FOR) Construir um programa que leia uma quantidade não determinada de pares de valores inteiros e positivos(*m,n*) , um par de cada vez. Em seguida deve ser mostrada a soma dos números inteiros entre *m* e *n*. A digitação dos pares termina quando *m* for maior ou igual a *n*.



### O Comando do-while

A terceira estrutura de repetição que veremos é o **do-while** de forma geral:

*do*

*{*

*<ação>;*   
*} while (condição);*

Mesmo que a <ação> seja apenas um comando é uma boa prática deixar as chaves. O uso do ponto-e- vírgula final é obrigatório. Vamos, como anteriormente, ver o funcionamento da estrutura **do-while** "por dentro":

*<ação>;*   
*if (<condição>) "Volta para a <ação>"*

Vemos pela análise do bloco acima que a estrutura **do-while** executa a <ação>, testa a <condição> e, se ela for verdadeira, volta para a <ação>. A grande novidade no comando **do-while** é que ele, ao contrário do [**for**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c430.html) e do [**while**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c440.html), garante que a <ação> será executada pelo menos uma vez.

Um dos usos da extrutura **do-while** é em menus, nos quais você quer garantir que o valor digitado pelo usuário seja válido, conforme apresentado abaixo: O menu será apresentado até que o usuário escolha uma das opções válidas.

#include <stdio.h>

#include <stdlib.h>

int main ()

{

int i;

do

{

printf ("\n\nEscolha 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));

switch (i)

{

case 1:

printf ("\t\tVoce escolheu Mamao.\n");

break;

case 2:

printf ("\t\tVoce escolheu Abacaxi.\n");

break;

case 3:

printf ("\t\tVoce escolheu Laranja.\n");

break;

}

system("PAUSE");

return(0);

}

EXERCÍCIOS

* 1. (USAR DO/WHILE e IF/ELSE) Escrever um programa que lê um número não conhecido de valores inteiros, um de cada vez, e conta quantos deles estão em cada um dos intervalos:

Intervalo 1🡪 1..25,

Intervalo 2🡪 26..50

Intervalo 3🡪51..75

Intervalo 4🡪76..100.

Parar de informar quando o número 0 for digitado. Mostrar quantos números foram contados em cada intervalo e quantos não se encaixaram em nenhum dos intervalos definidos.

### O Comando break

Nós já vimos o uso do comando **break** interrompendo os comandos [**switch**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c420.html). Ele pode também interromper a execução de um laço de repetição. Esses são os dois usos do comando **break**: ele pode quebrar a execução de um comando (como no caso do **switch**) ou interromper a execução de *qualquer* loop (como no caso do **for**, do **while** ou do **do while**). O **break** faz com que a execução do programa continue na primeira linha seguinte ao loop ou  bloco que está sendo interrompido.

Observe que um break causará uma saída somente do laço mais interno. Por exemplo:

#include <stdio.h>

#include <stdlib.h>

int main ()

{

int t, count;

for(t=0; t<=100; ++t)

{

count=1;

for(;;)

{

printf("%d", count);

count++;

if(count==10) break;

}

printf("\n");

}

system("PAUSE");

return(0);

}

O código acima imprimirá os números de 1 a 9 cem vezes na tela. Toda vez que o **break** é encontrado, o controle é devolvido para o laço for externo e o comando printf(“\n”); será executado.

Outra observação é o fato que um **break** usado dentro de uma declaração switch afetará somente os dados relacionados com o switch e não qualquer outro laço em que o switch estiver.

**O Comando continue**

O comando **continue** pode ser visto como sendo o oposto do [**break**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c460.html). Ele só funciona dentro de um laço de repetição. Quando o comando **continue** é encontrado, o laço pula para a próxima iteração, sem o abandono do laço, ao contrário do que acontecia no comando **break**. O programa abaixo exemplifica o uso do *continue*:

#include <stdio.h>

#include <stdlib.h>

int main()

{

int opcao;

while (opcao != 5)

{

printf("\n\n Escolha uma opção entre 1 e 5: ");

scanf("%d", &opcao);

if ((opcao > 5)||(opcao <1)) continue; /\* Opção invalida: volta ao inicio do loop \*/

switch (opcao)

{

case 1:

printf("\n --> Primeira opção...");

break;

case 2:

printf("\n --> Segunda opção...");

break;

case 3:

printf("\n --> Terceira opção...");

break;

case 4:

printf("\n --> Quarta opção...");

break;

case 5:

printf("\n --> Abandonando..\n");

break;

}

}

system("PAUSE");

return(0);

}

O programa acima ilustra uma aplicação simples para o *continue*. Ele recebe uma opção do usuário. Se esta opção for inválida, o *continue* faz com que o fluxo seja desviado de volta ao início do laço de repetição (loop). Caso a opção escolhida seja válida o programa segue normalmente.