Aula 05

Site: MoodleWIFI
Curso: Programação e Algoritmos
Livro: Aula 05
Impresso por: RIANE RUBIO
Data: Thursday, 11 Apr 2019, 19:21

Sumário

1. Introdução

Comandos de Repetição (Laços ou Loops)

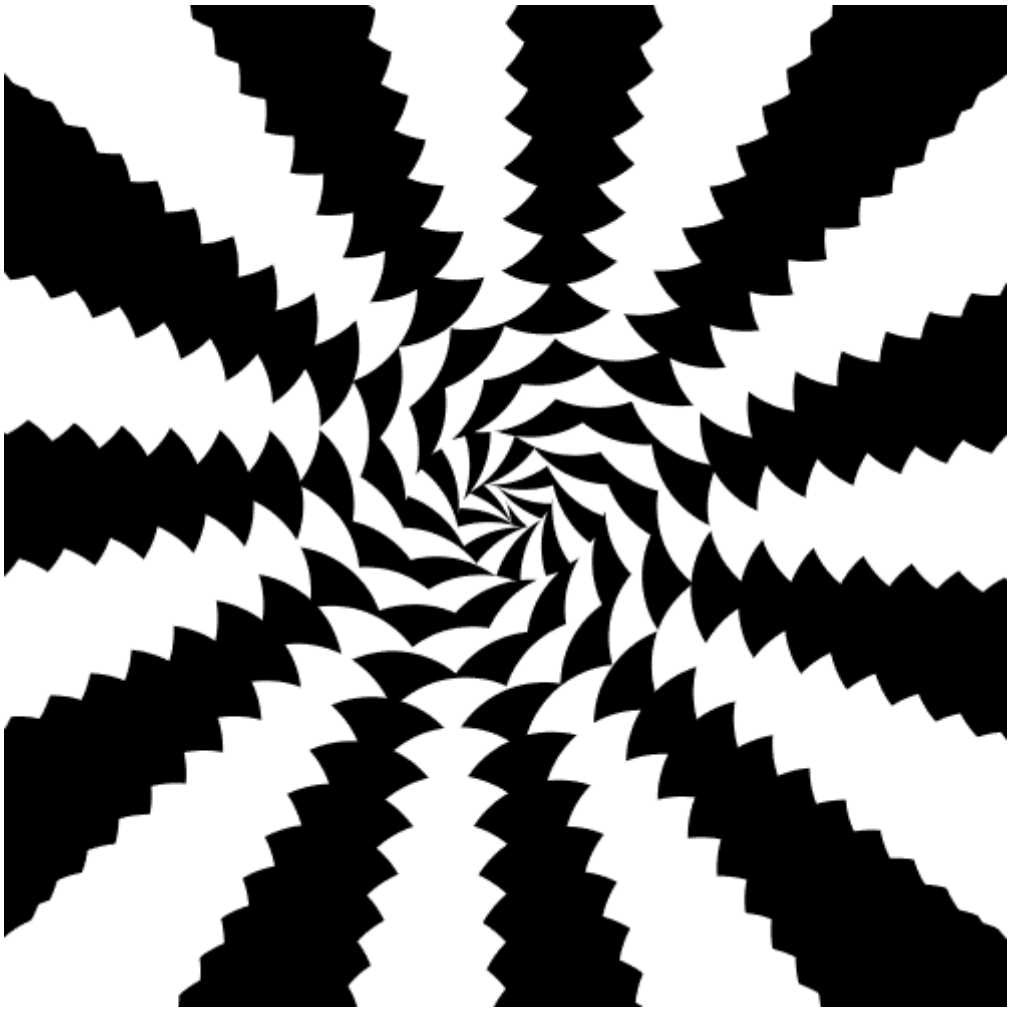
Nas últimas aulas foi visto como desviar o fluxo do programa.

Mas, e se for necessário realizar uma mesma tarefa muitas vezes seguidas?

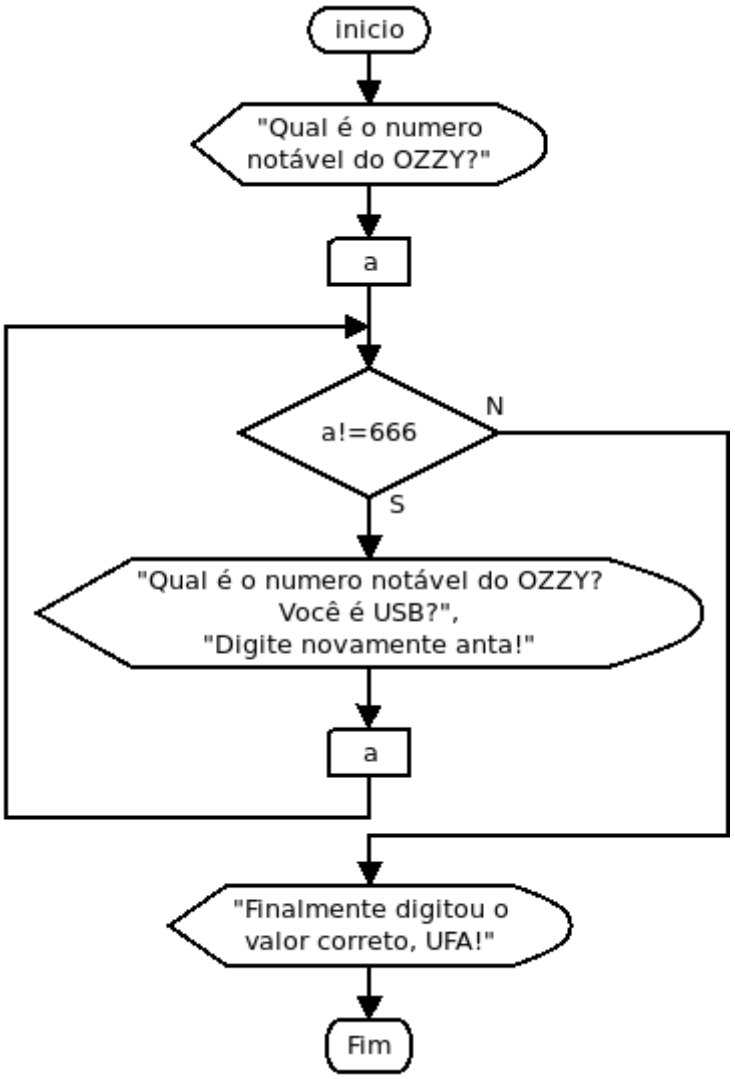
É aí que entram os loops.

Existem três tipos: **while**, **do while** e **for**;

Enquanto, **faça enquanto** e **para**



2. Tipos de loops
Loops condicionais

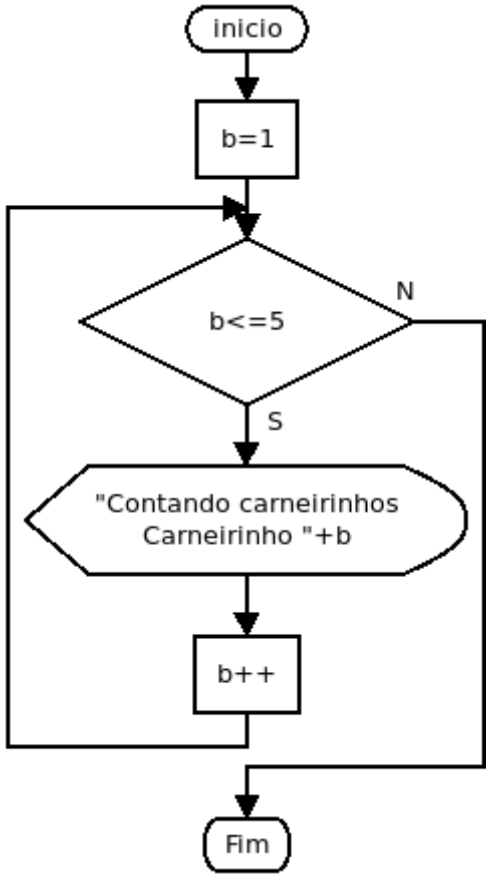


Exemplo1

Código-fonte:

```
<form>
<input type="button" value="Exemplo1" onclick="condic()">
</form>
<script>
function condic(){
a=parseInt(prompt("Qual é o numero notável do OZZY?","Digite aqui!"));
  while(a!=666)
  a=parseInt(prompt("Qual é o numero notável do OZZY?\nVocê é USB?","Digite novamente a
alert("Finalmente digitou o valor correto, UFA!");
}
</script>
```

Loops contados



Exemplo2

Código-fonte

```
<form>
<input type="button" value="Exemplo2" onclick="cont()">
</form>
<script>
function cont(){
b=1;
  while(b<=5)
  {
  alert("Contando carneirinhos\nCarneirinho "+b);
  b++;
  }
}
</script>
```

3. While (enquanto)

A sintaxe desse loop é a seguinte:

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

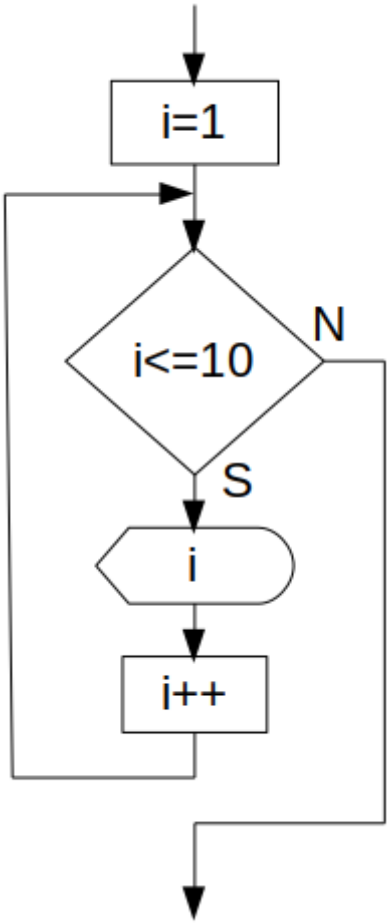
Esse código executa comandos enquanto expressão for verdadeira.
Quando expressão for falsa, o programa pula comandos e vai para depois do **while**;

Por exemplo, o seguinte programa escreve os números de 1 a 10:

1. ```
   #include <stdio.h>

   int main(){

       int i = 1;                  // Declaração e inicialização

       while(i <= 10){             // Expressão lógica
           printf("%d ", i);       // Comando 1: escreve o número na tela
           i++;                    // Comando 2: Incrementação do i
       }

       getchar();
       return 0;
   }
   ```

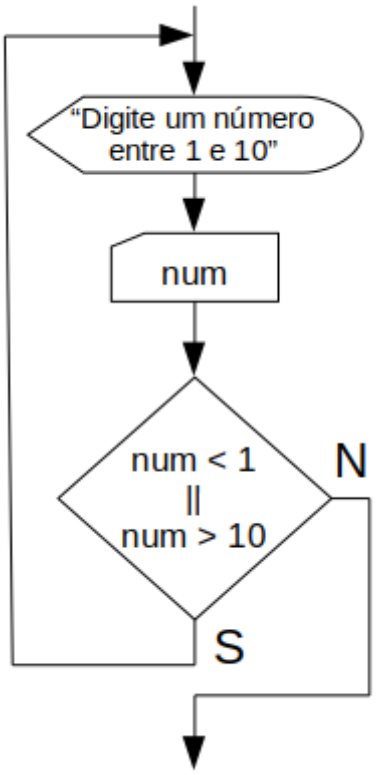
Saída (output):

```
1 2 3 4 5 6 7 8 9 10
```

Obs: não se esqueça de incrementar o i, senão o loop se torna **infinito**.
```


i=1
i<=10
N
S
i
i++

4. Do while (faça enquanto)

Sintaxe:

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

É bem parecido com o **while** loop, mas ele executa comandos pelo menos uma vez.

Por exemplo, o seguinte programa pede do usuário um número entre 1 e 10.
Enquanto ele não digitar um número válido, o programa vai perguntar de novo.

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

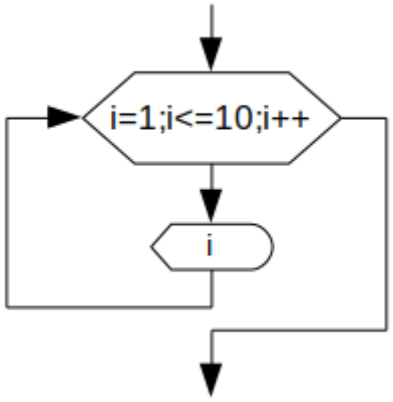
     int num; // Declaração

    do {
        printf("Digite um numero entre 1 e 10: "); // Comando 1: pedido ao usuário
        scanf("%d", &num);                         // Comando 2: armazenamento em num
       } while(num < 1 || num > 10);               // Expressão lógica

    getchar();
    return 0;
}
```

Exemplo de output (com input -3, 42 e 7):

Digite um numero entre 1 e 10: -3
Digite um numero entre 1 e 10: 42
Digite um numero entre 1 e 10: 7
Obs: não se esqueça do **ponto-e-vírgula** depois do **while**(expressão).


"Digite um número
entre 1 e 10"
num
num < 1
||
num > 10
N
S

5. For (para)

```
for (expr1; expr2; expr3) {
    comandos;
}
```

Parece mais complicado, mas é de longe o loop mais usado nos programas, em parte por ser mais sintético.
Ele é equivalente ao seguinte código com o **while** loop:

```
expr1;
while (expr2) {
    comandos;
    expr3;
}
```

Normalmente, temos as seguintes correlações:
- expr1: declarações e inicializações
- expr2: expressão lógica
- expr3: incrementação ou alteração de uma variável que está em expr2

Por exemplo, o seguinte código também imprime os números de 1 a 10 (*compare com o exemplo do **while***):

```
#include <stdio.h>

int main(){

    for (int i = 1; i <= 10; i++) {
        printf("%d ", i);
    }

    getchar();
    return 0;
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Obs: o escopo das variáveis que foram declaradas na expr1 é apenas dentro do **for**.



Particularidades:

- As expressões podem ser vazias

```
   for(char ch = 'n'; ch != 's'; ){       // Sem expr3
       printf("Deseja sair (s/n) ? ");
       rewind(stdin);                     // Para esvaziar o buffer
       scanf("%c", &ch);
   }
```

/* Ou até o seguinte loop infinito */

```
   for(;;){                           // Sem nenhuma
       printf("X");
   }
```

- As expressões podem ser compostas

```
       for(int i=0, j=9; i < 10; i++, j--){     // expr1 e expr3 são compostas
           printf("%d - %d\n", i, j);
       }
```

- As expressões podem ser outros comandos

```
       for(char ch = 'n'; ch != 's'; scanf("%c", &ch)){
           printf("Sair? ");
           rewind(stdin);
```

}

6. Break

Essa palavra-chave faz com que o loop em que ela está termine.
Se houver **nested loops**, apenas o loop mais interno será terminado.

Por exemplo, retomemos o loop infinito do **for,** mas com um meio de sair do loop, clicando qualquer tecla:

1. ```
   #include <stdio.h>
   #using <mscorlib.dll>

   using namespace System;

   int main(){

       for(;;){
           printf("X");
           if(Console::KeyAvailable)
               break;
       }

       getchar();
       return 0;
   }
   ```

No programa acima, o loop roda continuamente.
Quando alguém apertar alguma tecla, **Console::KeyAvailable** retornará **true**, e o comando **break** será executado.
Ou seja, o programa sairá do loop.
   ```

7. Continue

Esse comando faz com que a execução do programa pule para o **topo do loop**.

Por exemplo, o seguinte código imprime todos os números de 0 a 50, exceto os que são múltiplos de 3 ou 4:

```
#include <stdio.h>

int main(){

    for(int i=0; i<=50; i++){
        if(i%3 == 0 || i%4 == 0)
            continue;
        printf("%d ", i);
    }

    getchar();
    return 0;
}
```

Output:

```
1 2 5 7 10 11 13 14 17 19 22 23 25 26 29 31 34 35 37 38 41 43 46 47 49 50
```

Obs: Apesar de ter pulado o final do **for**, a variável i foi incrementada.

Tome cuidado com o **while** e o continue, porque podem ocorrer loops infinitos:

```
1. #include <stdio.h>
2.
3. int main(){
4.     int i=1;
5.     while(i < 10){
6.         if(i == 5)
7.             continue;
8.         printf("%d ", i);
9.         i++;
10.    }
11.
12.    getchar();
13.    return 0;
14. }
```

O programa acima fica preso indefinidamente nas linhas 5, 6 e 7, a partir do momento em que i = 5;

Aviso: os professores e livros de programação normalmente aconselham **evitar** o uso de **break** e **continue**, já que o código fica mais difícil de acompanhar (pula de um lado para o outro).

8. Nested loops (loops aninhados)

É possível colocar loops uns dentro dos outros, para realizar tarefas repetitivas dentro de outras tarefas repetitivas.
Só cuidado com o escopo dos contadores (i, j, ...) .

No seguinte código há um **for** dentro de outro **for**:

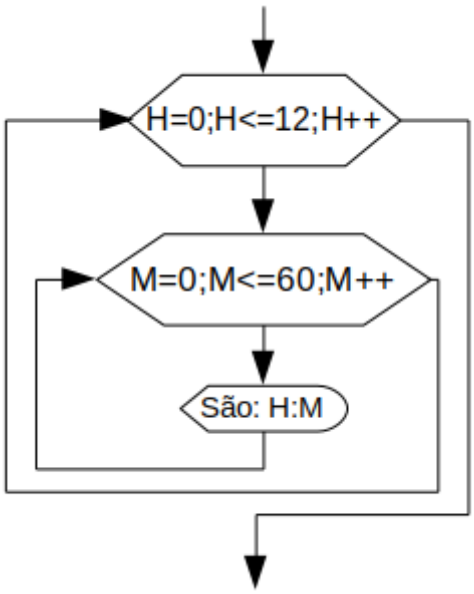
```
#include <stdio.h>
#include <math.h>
#using <mscorlib.dll>

using namespace System;

int main(){
    // Mude este número se no seu computador estiver muito rápido ou muito lento
    const int tempo = 70;

    // Definindo a constante 'PI'.
    // A palavra 'const' faz com que essa variável não possa ser alterada depois
    const double PI = 3.14159265358979323846264338;

    // Vamos iterar ( = realizar ciclos ) enquanto o usuário não clicar em nenhuma
tecla
    for(int i=0; !Console::KeyAvailable; i++){
        // A partir de 'i', calcula-se quantas vezes o " " será impresso
        for(int j = 0; j < 40 + 25*sin(PI/15*i); j++){
            printf(" ");
        }

        // Esse asterísco é o que vocês realmente veêm
        printf("*\n");

        // Pára o programa por 'tempo' milisegundos
        Threading::Thread::Sleep(tempo);
    }

    return 0;
}
```

Output:

```
veja por si mesmo!      (dessa vez pode copiar...)
```

Obs: - para sair do programa, aperte qualquer tecla

A função Sleep(tempo) faz com que o programa espere *tempo* ms antes de continuar


H=0;H<=12;H++
M=0;M<=60;M++
São: H:M