

EAJ0365-Programação de computadores

Prof. Dr. Josenalde Barbosa de Oliveira

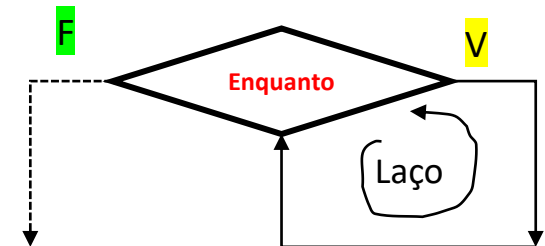
josenalde.oliveira@ufrn.br

Estrutura de controle de fluxo: *repetição...*

- Usado para modelar ações que precisam ser mantidas/repetidas **enquanto** determinadas condições ou critérios lógicos são **verdadeiros (=1)**
- As estruturas em pseudocódigo são 03: **faça...enquanto**, **enquanto....faça** e **para.....faça** e permitem a criação de LAÇOS de repetição (LOOPS)
- Laços podem ser simples ou aninhados/compostos (*nested loop*)
- Necessário definir critério de parada e variável de controle (contador, iterador)
- O teste lógico é realizado no início, ou seja, se já for Falso, não entra no laço, não é executado nenhuma vez o blocoComandosSeVerdadeiro
- Em Javascript: SIMPLES ENQUANTO (while)

```
inicializaContador()  
while (<condicaoLogica>){  
    <blocoComandosSeVerdadeiro>  
    atualizaContador()  
}  
<blocoComandosSeFalso>
```

Avaliado como V ou F



Enquanto....faça (repita) - while

Algoritmo loopInfinito

Início

Enquanto(**1**) Faça
 Exiba("info")

Fim_Enquanto

Fim

Algoritmo loopInfinito2

Início

N <- 4 → Inicialização da var. controle N

Enquanto(**N mod 2 == 0**) Faça

 Exiba("info")

Fim_Enquanto

Fim

↔ N % 2

► Em ambos os algoritmos, o critério de parada é sempre Verdadeiro, pois no primeiro caso não há variável de controle e no segundo caso a mesma não é alterada/atualizada dentro do laço, de modo a tornar o critério de parada Falso.

► No caso ao lado, embora o Incremento na variável N mantenha o critério sempre Verdadeiro, há um teste lógico que força a saída do laço, com o comando **break**.

```
var N = 4; //loop1.js
while (N % 2 == 0) {
    if (N > 20) break;
    console.log("01");
    N += 2;
}
console.log("Conseguir sair...com N=%d",N);
```

Enquanto....faça (repita) - while

```
var N = 4, iter=0; //loop2.js
while (N % 2 == 0) {
  console.log("N= %d " + "iter = %d", N, iter);
  //exibir(linhaTabela)

  if (N > 20) break;
  console.log("info ");
  N += 2; iter++;
}
console.log("N= %d " + "iter = %d", N, iter);
```

Iteração	N	N%2==0?	N>20?	Out
0	4	V	F	info
1	6	V	F	info
2	8	V	F	info
3	10	V	F	info
4	12	V	F	info
6	14	V	F	info
7	16	V	F	info
8	18	V	F	info
9	20	V	F	info
10	22	V	V	N=22

➡ É muito importante o debug do Código, acompanhar a evolução das variáveis para verificar eventuais erros semânticos! (teste de mesa ou uso do debugger)

➤ Qual a saída do Código 1 abaixo

➤ se A=1 e B=13:

```
var scanf = require('scanf');
var S = 0, A, B, i; // loop3.js
[A, B] = scanf("%d %d");
i = A;
//entre A e B, inclusive
while (i <= B) {
    if (i % 2 != 0) S += i;
    i++;
}
console.log("S = ", S);
```

Problema clássico: SOMATÓRIO

$$\sum_{i=A}^B i, \text{ se } i \text{ é ímpar}$$

i	i<=B?	i%2!=0	S
1 (A)	V	V	S=0+1=1
2	V	F	1
3	V	V	S=1+3=4
4	V	F	4
5	V	V	S=4+5=9
6	V	F	9
7	V	V	S=9+7=16
8	V	F	16
9	V	V	S=16+9=25
10	V	F	25
11	V	V	S=25+11=36
12	V	F	36
13	V	V	S=36+13=49
14	F	-	49

➤ Qual a saída do Código 2 abaixo

➤ se A=1 e B=13:

```
var P = 1, A, B, i; //loop4.js
var scanf = require('scanf');
[A, B] = scanf("%d %d");
i = A;
//entre A e B, inclusive
while (i <= B) {
    if (i % 2 == 0) P *= i;
    i++;
}
console.log("P= ", P);
```

Problema clássico: PRODUTÓRIO

$$\prod_{i=A}^B i, \text{ se } i \text{ é par}$$

i	i<=B?	i%2==0	P
1 (A)	V	F	1
2	V	V	P=1*2=2
3	V	F	2
4	V	V	P=2*4=8
5	V	F	8
6	V	V	P=8*6=48
7	V	F	48
8	V	V	P=48*8=384
9	V	F	384
10	V	V	P=384*10=3840
11	V	F	3840
12	V	V	P=3840*12=46080
13	V	F	46080
14	F	-	46080

While...análise de fluxograma

► Qual a saída do fluxograma ao lado para $M = 40$? E para $M = 15$?

Algoritmo fluxo1

Início

Leia(M)

NM = M

C = 5

Enquanto(C!=15) Faça

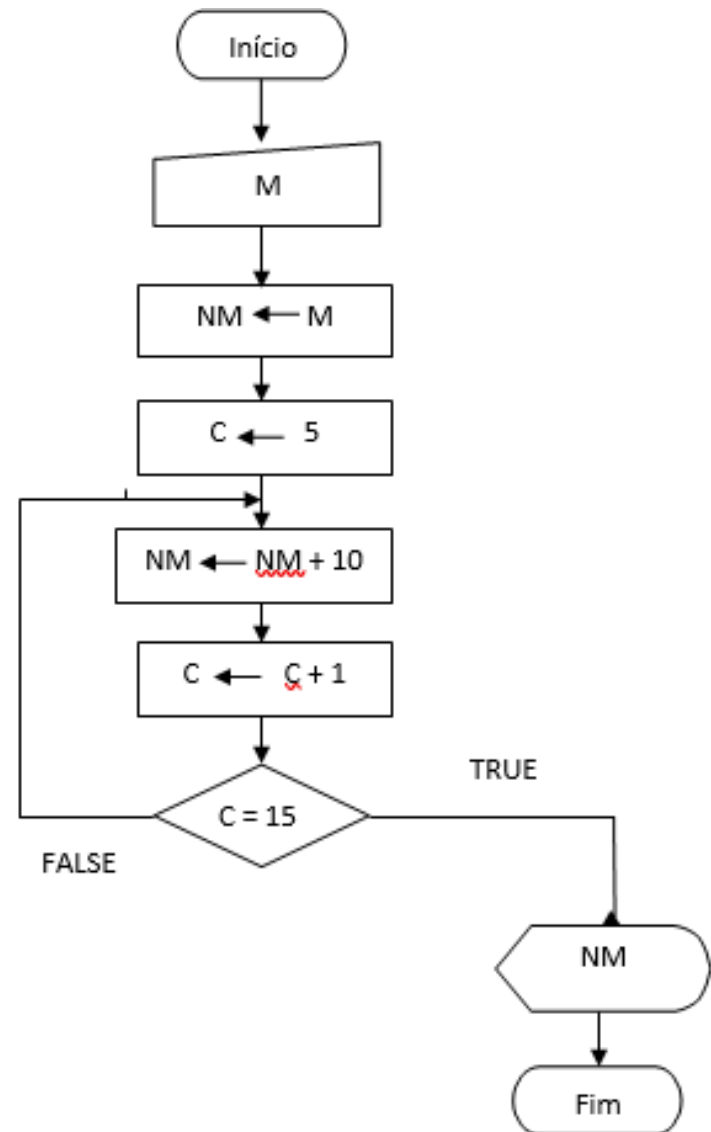
 NM = NM + 10

 C = C + 1

Fim_Enquanto

Exiba(NM)

Fim

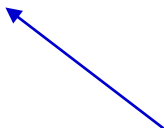


Exercício: aproximação do π

- ▶ O número PI pode ser calculado (aproximado) por meio da série infinita:

$$\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \dots \right)$$

- ▶ Escreva o código em Javascript que calcule e exiba o valor do número PI, utilizando a série anterior, até que o valor absoluto (modulo) da diferença entre o número calculado numa iteração e o da anterior seja menor ou igual a 5×10^{-12}




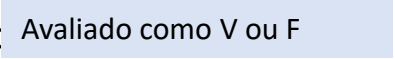
Critério de parada

Estrutura de controle de fluxo:

Faça...enquanto (do...while)

► Similar ao Enquanto (while), porém a condição ou critério de parada é testada apenas ao fim da primeira execução, ou seja, garante ao menos uma execução do **blocoComandosSeVerdadeiro**.

► Em Javascript: do {<blocoComandosSeVerdadeiro>}while <condição>;

```
inicializaContador()  
do {  
    <blocoComandosSeVerdadeiro>  
    atualizaContador()   
} while (<condicaoLogica>  Avaliado como V ou F  
<blocoComandosSeFalso>
```

Faça...enquanto (do...while)

Algoritmo doWhile

Início

i <- 0

Faça

Exiba("info")

i <- i + 1;

Enquanto(i <> 1)

Fim_FaçaEnquanto

Exiba("sai do laço")

```
var i = 0; //loop5.js
```

```
do {
```

```
    console.log("info ");
```

```
    i++;
```

```
} while (i != 1)
```

```
console.log("sai do laço");
```

➡ Muito usado para repetir testes, sem necessidade de estar recompilando a cada teste

Faça...enquanto (do...while)

```
var c,r; // loop6.js
do {
    console.log("Digite um caracter para saber o ascii associado: ");
    c = scanf("%c");
    console.log(c.charCodeAt(0));
    console.log("quer repetir (s/n)): ");
    r = scanf("%c");
} while(r != 'n' && r != 'N');
```

Para....faça (for)

➡ Comando que inclui a inicialização do contador, critério de parada e atualização do contador numa única linha, com a sintaxe:

➡ Em Javascript:

```
for ([inicializacaoContador];[critérioParada];[atualizacaoContador]) {  
    <blocoComandosAREpetir>  
}
```

```
var i, n = 0;  
for (i=0;i<5;i++) {  
    n += i;  
}  
console.log(n);  
console.log(i);  
/* neste caso, o contador i é  
visível fora do laço */
```

```
n = 0;  
for (var i=0; i<5; i++) n += i;  
console.log(n);  
console.log(i);
```

```
/* erro, pois a variável i foi declarada  
para uso apenas no for, escopo local ao  
laço. */
```

Para....faça (for)

► Casos especiais

```
var i = 0, n = 0;
for (; i<5; i++) {
    n += i;
}
/* neste caso, i já foi
inicializado fora do laço,
então não é necessário no for
*/
```

```
var n = 0;
for (int i=0; i<5; ) {
    n += i;
    i++;
}
```

```
var n = 0;
for (int i=0; ; i++) {
    n += i;
    if (i > 5) break;
}
/* loop infinito se não inserir condição
dentro do laço para sair */
```

```
for (var i=5, j=0; i>=0,j<6; i--,j++) {
    console.log(i + " " + j);
}
```

Para....faça (for)

► Casos especiais (combinando comandos)

Saída:

```
var i = 5;
while (i > 0) {
    console.log("i: ", i);
    for (var j=1; j <= i; j++) {
        console.log("j: \t", j);
    }
    i--;
}
console.log("fora do loop i = ", i);
```

```
i: 5
j:1 j:2 j:3 j:4 j:5
i: 4
j:1 j:2 j:3 j:4
i: 3
j:1 j:2 j:3
i: 2
j:1 j:2
i: 1
j:1
i fora do loop: 0
```

Uso do continue

- ➡ O comando continue – pula (skip) iteração atual e já atualiza o contador

```
var N = 6; //loop9.js
for (var i=0;i<N;i++) {
  if (i==4) continue;
  console.log(i);
}
```

Saída:

```
i: 0
i: 1 Quando i=4, não executa os comandos abaixo do
i: 2 continue, e já atualiza (no caso i++) o contador, e
i: 3 depois testa a condição de repetição.
i: 5
```

```
var N = 6;
for (var i=0;i<N;i++) {
  if (i>4) continue;
  console.log(i);
}
```

Saída:

```
i: 0
i: 1 Neste caso, equivale ao break, pois não
i: 2 executa os comandos para i > 4
i: 3
i: 4
```

Laços aninhados (compostos)

➡ É muito comum necessitar usar duas ou mais estruturas de repetição aninhadas, como por exemplo, com o **for**, em que surgem conceitos de laço externo e laço interno.

➡ Os laços são resolvidos (executados) do mais interno para o externo, ou seja, para cada iteração do laço externo (i), são executadas as iterações do laço interno (j), até que a condição interna seja falsa. Quando isto ocorre, a variável j é incrementada (atualizada) e o controle é devolvido ao laço externo que executa nova iteração para a variável de controle i.

```
for (var i=2; i<=9; i++) {  
    console.log("Tabuada do " + i);  
    for (var j=1; j<=9; j++) {  
        console.log(i + " x " + j + " = " + i*j);  
    }  
} // loop10.js
```

Laço externo: i

Laço interno: j

Laços aninhados (compostos)

```
var i,j;
for (i=1;i<5;i++) {
  for (j=1;j<5;j++) {
    if (i==j) console.log(" 1");
    else console.log(" 0");
  }
  console.log("\n");
} //loop11.js
```

No caso do for, a cada volta ao laço externo, o laço interno é reinicializado

```
var i=1, j;
while (i<5) {
  j = 1;
  while (j<5) {
    if (i==j) console.log(" 1");
    else console.log(" 0");
    j++;
  }
  console.log("\n");
  i++;
}
```

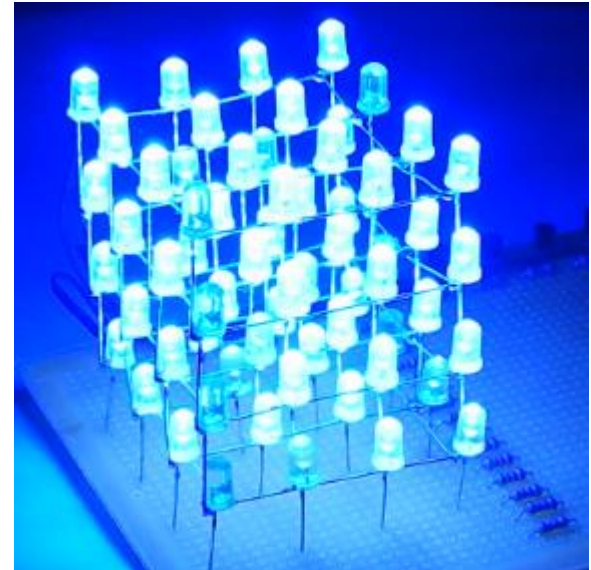
No caso do while, a cada volta ao laço externo, é preciso reiniciar explicitamente a variável de controle do laço interno, no caso ao lado, j.

```
var i,j,k;
for (i=0;i<5;i++)
  for (j=0;j<5;j++)
    for (k=0;k<5;k++)
      if (i==2 && j==3 && k==1) break;
      else console.log("Ponto " + i + j + k + "no cubo");

console.log("FIM");
```

<https://www.youtube.com/watch?v=K0xfp9yTToA>

Exemplo 3D: cubo de leds



Exercícios propostos

1. Faça um programa que calcule e mostre a soma dos 50 primeiros números pares.
2. Faça um programa que verifique se um número natural fornecido pelo usuário é primo.
3. Faça um programa que leia um número natural e informe seus divisores. Por exemplo: $66 = 1, 2, 3, 6, 11, 22, 33, 66$.
4. Escreva um programa que gere os N termos da série de Fibonacci: 0,1,1,2,3,5,8,13,21,34, ..., ou seja, a partir do 4. termo da série, seu valor é a soma dos dois valores anteriores (sem usar vetores).
5. Faça um programa que leia vários números inteiros até que se digite um número negativo. Quando isto ocorrer, mostrar na tela o maior e o menor número digitado pelo usuário.

Exercícios propostos

6. Escreva um programa que leia um número inteiro positivo N e em seguida imprima N linhas do chamado triângulo de Floyd: por exemplo, para N = 4:

1

2 3

4 5 6

7 8 9 10

....

7. Faça um programa que leia um natural N e mostre o valor de E, dado por

$$E = 1/1! + 1/2! + \dots + 1/N!$$