



Computação Eletrônica

Laços de repetição – for, while, do-while

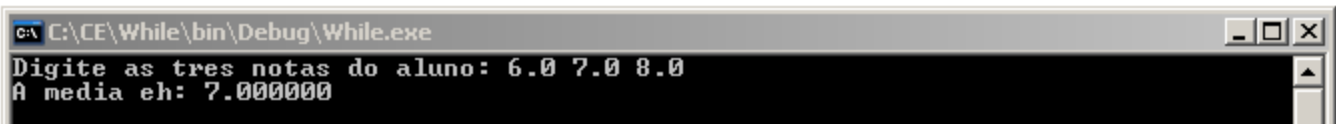
Prof: Luciano Barbosa



Estruturas de Repetição

- Imagine um programa que calcula a média de um aluno.
- Este programa é bastante simples, bastaria:
 - Ler as notas do teclado;
 - Calcular a média; e
 - Imprimir o resultado.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float nota1, nota2, nota3, media;
6      //Lê as notas
7      printf("Digite as tres notas do aluno: ");
8      scanf("%f %f %f", &nota1, &nota2, &nota3);
9      //Calcula a média
10     media = (nota1 + nota2 + nota3)/3;
11     //Imprime o resultado
12     printf("A media eh: %f\n", media);
13 }
14
```





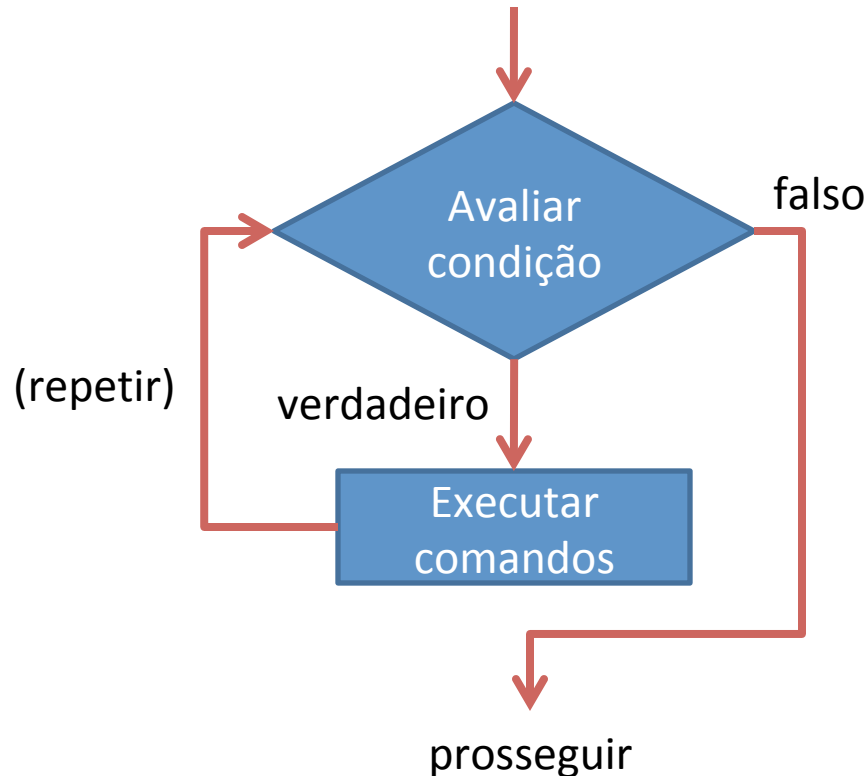
Estruturas de Repetição

- Como poderíamos fazer para calcular a média de cada um dos alunos de uma turma em um mesmo programa?
- Precisamos repetir os mesmos comandos, várias vezes, até que a média de cada aluno tenha sido calculada e impressa.
- Solução: **Laços** (em inglês: *Loops*)
- Em C temos três opções de estruturas de repetição:
 - **while**
 - **do – while**
 - **for**



Estruturas de Repetição - **while**

- Fluxograma do **while**:



- Há a possibilidade de nunca se executar os comandos caso a primeira avaliação da condição já resulte em falso.



Estruturas de Repetição - **while**

- A estrutura **while** possui a seguinte sintaxe:

```
while (expressaoBoleana) {  
    comando1;  
    comando2;  
    // ...  
}
```

- A palavra “*while*” significa “enquanto” em português, portanto, lê-se:
 - “Enquanto a expressão booleana for verdadeira, execute os comandos do bloco abaixo”.
 - Ou seja, o bloco de comandos será repetido enquanto a expressão booleana for verdadeira.
- Algo dentro do laço deve ser capaz de modificar o resultado da expressão booleana, caso contrário o laço nunca terminará, e o programa entrará em “**loop infinito**”.



Estruturas de Repetição - **while**

- Com o **while**, o programa anterior poderia ser alterado para trabalhar com vários alunos:

Variável **repetir**:

valor 1: o programa deve continuar;

valor 0: o programa deve terminar.

O valor de **repetir** pode mudar para zero na linha 21 caso o caractere lido não seja 's', o que finaliza o laço.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float nota1, nota2, nota3, media;
6      int repetir=1;
7      char digito;
8
9      while (repetir) {
10         //Lê as notas
11         printf("Digite as tres notas do aluno: ");
12         scanf("%f %f %f", &nota1, &nota2, &nota3);
13         //Calcula a média
14         media = (nota1 + nota2 + nota3)/3;
15         //Imprime o resultado
16         printf("A media eh: %f\n", media);
17         //Deve continuar?
18         printf("continuar? (s/n):");
19         fflush(stdin);
20         scanf("%c", &digito);
21         if (digito!='s') repetir = 0;
22     }
23 }
```

```
C:\CE\While\bin\Debug\While.exe
Digite as tres notas do aluno: 7.0 8.0 9.0
A media eh: 8.000000
continuar? (s/n):s
Digite as tres notas do aluno: 9.5 9.0 8.5
A media eh: 9.000000
continuar? (s/n):s
Digite as tres notas do aluno: 5.5 6.5 7.5
A media eh: 6.500000
continuar? (s/n):n
```



Estruturas de Repetição - **while**

- Variações do **while**:

```
while (expressaoBoleana) {  
    comando1;  
    comando2;  
    // ...  
}
```

```
while (expressaoBoleana)  
    comando;
```

```
while (expressaoBoleana) comando;
```

- As chaves são necessárias apenas quando há mais de um comando a ser repetido.



Estruturas de Repetição - **while**

- Exemplos de uso do **while**:

```
int i=0;

while (i<10) {
    printf("i: %d\n", i);
    i++;
}
```

```
int counter = 5;
long factorial = 1;

while (counter > 1)
{
    factorial *= counter--;
}
printf("%d", factorial);
```

```
#define FALSE 0
#define TRUE 1

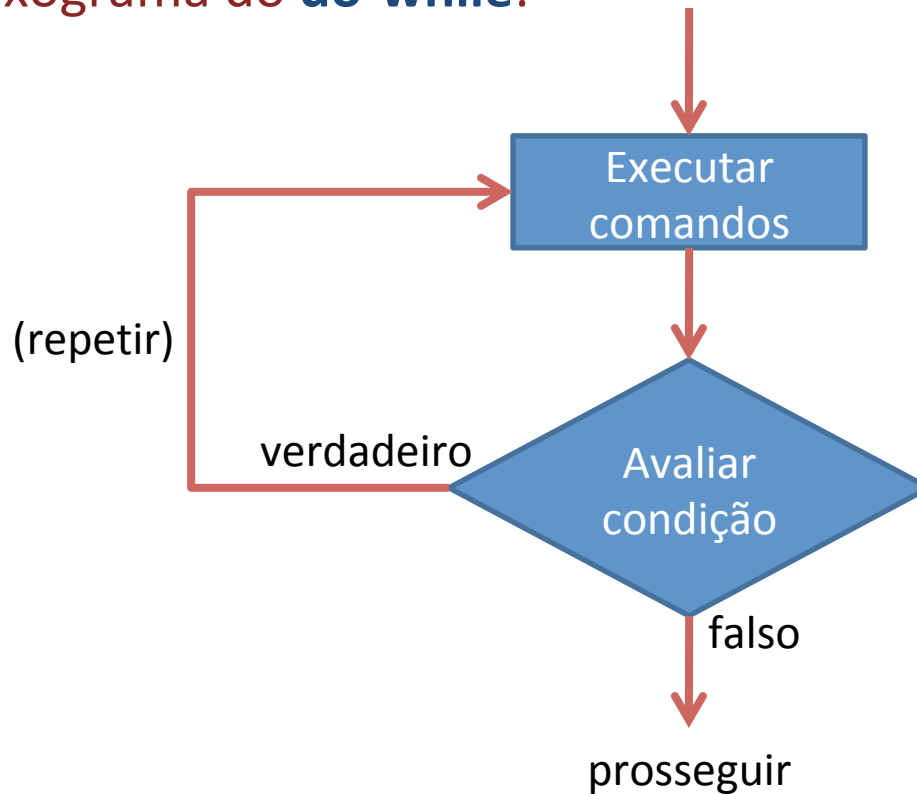
int terminar=FALSE;
int i=0;

while (!terminar) {
    printf("Continuando...\n");
    if (i>10) terminar = TRUE;
    i++;
}
```



Estruturas de Repetição: **do-while**

- Fluxograma do **do-while**:



- Os comandos serão executados ao menos uma vez, pois a avaliação da condição ocorre apenas após a primeira execução.



Estruturas de Repetição: **do-while**

- A estrutura **do-while** possui a seguinte sintaxe:

```
do {  
    comando1;  
    comando2;  
    // ...  
} while (expressaoBoleana);
```

- A diferença do **while** para o **do-while** é que, no **do-while** sempre acontece a primeira execução do bloco de comandos e a expressão booleana só é avaliada ao final de cada execução.
 - Se for **verdadeira**, o bloco de comandos é repetido;
 - Se for **falso**, o laço termina.



Estruturas de Repetição: **do-while**

- Com o **do-while**, o programa anterior poderia ser escrito assim:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float nota1, nota2, nota3, media;
6      char digito;
7
8      do {
9          //Lê as notas
10         printf("Digite as tres notas do aluno: ");
11         scanf("%f %f %f", &nota1, &nota2, &nota3);
12         //Calcula a média
13         media = (nota1 + nota2 + nota3)/3;
14         //Imprime o resultado
15         printf("A media eh: %f\n", media);
16         //Deve continuar?
17         printf("continuar? (s/n):");
18         fflush(stdin);
19         scanf("%c", &digito);
20     } while (digito == 's');
21 }
```

Podemos utilizar expressões booleanas tão complexas quanto se queira dentro dos parênteses do **while**.

Obs.: Zero sempre é interpretado como falso e qualquer outro número como verdadeiro.

```
C:\CE\While\bin\Debug\While.exe
Digite as tres notas do aluno: 7.0 8.0 9.0
A media eh: 8.000000
continuar? (s/n):s
Digite as tres notas do aluno: 9.5 9.0 8.5
A media eh: 9.000000
continuar? (s/n):s
Digite as tres notas do aluno: 5.5 6.5 7.5
A media eh: 6.500000
continuar? (s/n):n
```



Estruturas de Repetição: **do-while**

- Variações do **do-while**:

```
do {  
    comando1;  
    comando2;  
    // ...  
} while (expressaoBoleana);
```

```
do  
    comando;  
while (expressaoBoleana);
```

```
do comando; while (expressaoBoleana);
```

- As chaves são necessárias apenas quando há mais de um comando a ser repetido.



Estruturas de Repetição - do-while

- Exemplos de uso do **do-while**:

```
int i=0;

do {
    printf("i: %d\n", i);
    i++;
} while (i<10);
```

```
#define FALSE 0
#define TRUE 1

int terminar=FALSE;
int i=0;

do {
    printf("Continuando...\n")
    if (i>10) terminar = TRUE;
    i++;
} while (!terminar);
```

```
int counter = 5;
long factorial = 1;

do
{
    factorial *= counter--;
} while (counter > 1);
printf("%d", factorial);
```



Estruturas de Repetição: Contadores

- Algumas vezes sabemos exatamente quantas vezes queremos executar os comandos.
- Nesses casos podemos utilizar uma variável inteira como contador, incrementando-a a cada execução, e verificando se a mesma atingiu valor desejado de execuções.
- Ex.: Imprimir todos os números de 1 a 15:

```
1 | #include <stdio.h>
2 |
3 | int main()
4 | {
5 |     int i=1;
6 |
7 |     while(i<=15) {
8 |         printf("Execucao %d\n", i);
9 |         i++;
10 |     }
11 |
12 |     printf("Fim.\n");
13 | }
14 |
```

Annotations on the left:

- Inicialização** (points to line 5: `int i=1;`)
- Avaliar condição** (points to line 7: `while(i<=15)`)
- incremento** (points to line 9: `i++;`)

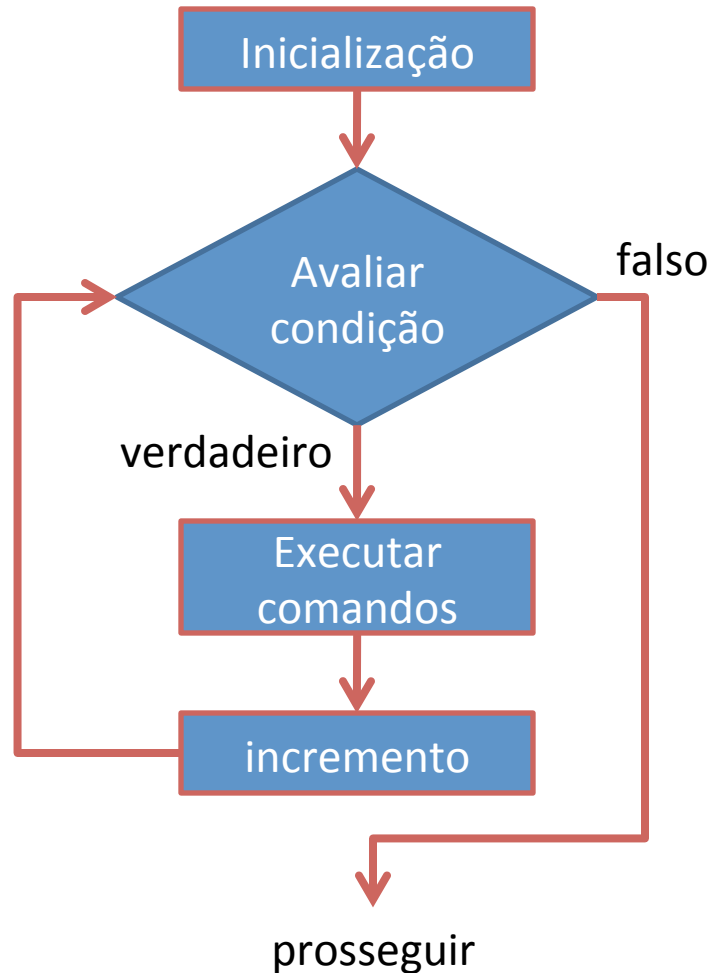
Terminal output (C:\CE\For\bin\Debug\For.exe):

```
Execucao 1
Execucao 2
Execucao 3
Execucao 4
Execucao 5
Execucao 6
Execucao 7
Execucao 8
Execucao 9
Execucao 10
Execucao 11
Execucao 12
Execucao 13
Execucao 14
Execucao 15
Fim.
```



Estruturas de Repetição - **for**

- Fluxograma do **for**:





Estruturas de Repetição: **for**

- A estrutura **for** é especializada para estes casos.
- Sua sintaxe é a seguinte:

```
for (inicializacao; expressaoBoleana; incrementos) {  
    comando1;  
    comando2;  
    // ...  
}
```

- No **for** há locais específicos para se definir:
 - A **inicialização** (executada apenas antes do primeiro laço);
 - A **expressão booleana** (avaliada sempre no começo de cada laço);
 - Os **incrementos** (realizados apenas ao final de cada laço);
- Obs.: nenhum destes termos é obrigatório.
- Melhora a legibilidade do código, pois agrupa em um mesmo local a inicialização, o incremento a ser dado e a condição de término do laço.



Estruturas de Repetição: for

- Se soubermos de antemão quantos alunos há na turma (por exemplo: 5) , o programa anterior poderia ser escrito assim:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float nota1, nota2, nota3, media;
6      int conta;
7
8      for (conta=1; conta<=5; conta++) {
9          //Lê as notas
10         printf("Digite as tres notas do aluno %d: ", conta);
11         scanf("%f %f %f", &nota1, &nota2, &nota3);
12         //Calcula a média
13         media = (nota1 + nota2 + nota3)/3;
14         //Imprime o resultado
15         printf("A media eh: %f\n", media);
16     }
17     printf("Fim.");
18 }
19
```

```
C:\CE\For\bin\Debug\For.exe
Digite as tres notas do aluno 1: 5.0 4.5 8.0
A media eh: 5.833333
Digite as tres notas do aluno 2: 8.5 9.0 7.5
A media eh: 8.333333
Digite as tres notas do aluno 3: 4.5 2.3 5.0
A media eh: 3.933333
Digite as tres notas do aluno 4: 4.0 5.0 6.0
A media eh: 5.000000
Digite as tres notas do aluno 5: 7.5 6.5 8.0
A media eh: 7.333333
Fim.
```



Estruturas de Repetição: **for**

- Variações do **for** (nem todas são recomendáveis):

```
for (inicializacao; expressaoBoleana; incrementos) comando;
```

```
inicializacao;  
for (; expressaoBoleana; incrementos) {  
    comandos;  
}
```

```
inicializacao;  
for (; expressaoBoleana; ) {  
    comandos;  
    incrementos;  
}
```

```
for (;;) {  
    //loop infinito  
    comandos;  
}
```

- As chaves são necessárias apenas quando há mais de um comando a ser repetido.



Estruturas de Repetição: **for**

- Outros exemplos de uso do **for**:

```
int i;  
  
for (i=99; i>0; i-=2) {  
    printf("i:%d\n", i);  
}
```

```
int i=0;  
  
for (; i<100; i=i+2) {  
    printf("i:%d\n", i);  
}
```

```
int i, j;  
  
for (i=0, j=100; i<40 || j>60; i++, j--) {  
    printf("i:%d j:%d\n", i, j);  
}
```



Estruturas de Repetição: for

- Outros exemplos de uso do **for**:

```
1  #include <stdio.h>
2  int main()
3  {
4      int i;
5      //Imprime toda a tabela ASCII:
6      for (i=0; i<256; i++) {
7          printf("O codigo ASCII de '%c' eh %d\n", i, i);
8      }
9  }
```

C:\CE\For\bin\Debug\For.exe

0	codigo	ASCII	de	' '	eh	0
0	codigo	ASCII	de	'@'	eh	1
0	codigo	ASCII	de	'A'	eh	2
0	codigo	ASCII	de	'B'	eh	3
0	codigo	ASCII	de	'C'	eh	4
0	codigo	ASCII	de	'D'	eh	5
0	codigo	ASCII	de	'E'	eh	6
0	codigo	ASCII	de	'F'	eh	7
0	codigo	ASCII	de	'G'	eh	8
0	codigo	ASCII	de	'H'	eh	9
0	codigo	ASCII	de	'I'	eh	10
0	codigo	ASCII	de	'J'	eh	11
0	codigo	ASCII	de	'K'	eh	12
0	codigo	ASCII	de	'L'	eh	13
0	codigo	ASCII	de	'M'	eh	14
0	codigo	ASCII	de	'N'	eh	15
0	codigo	ASCII	de	'O'	eh	16
0	codigo	ASCII	de	'P'	eh	17
0	codigo	ASCII	de	'Q'	eh	18
0	codigo	ASCII	de	'R'	eh	19
0	codigo	ASCII	de	'S'	eh	20
0	codigo	ASCII	de	'T'	eh	21
0	codigo	ASCII	de	'U'	eh	22
0	codigo	ASCII	de	'V'	eh	23

C:\CE\For\bin\Debug\For.exe

0	codigo	ASCII	de	'W'	eh	32
0	codigo	ASCII	de	'X'	eh	33
0	codigo	ASCII	de	'Y'	eh	34
0	codigo	ASCII	de	'Z'	eh	35
0	codigo	ASCII	de	'['	eh	36
0	codigo	ASCII	de	']'	eh	37
0	codigo	ASCII	de	'^'	eh	38
0	codigo	ASCII	de	'_'	eh	39
0	codigo	ASCII	de	'`'	eh	40
0	codigo	ASCII	de	'a'	eh	41
0	codigo	ASCII	de	'b'	eh	42
0	codigo	ASCII	de	'c'	eh	43
0	codigo	ASCII	de	'd'	eh	44
0	codigo	ASCII	de	'e'	eh	45
0	codigo	ASCII	de	'f'	eh	46
0	codigo	ASCII	de	'g'	eh	47
0	codigo	ASCII	de	'h'	eh	48
0	codigo	ASCII	de	'i'	eh	49
0	codigo	ASCII	de	'j'	eh	50
0	codigo	ASCII	de	'k'	eh	51
0	codigo	ASCII	de	'l'	eh	52
0	codigo	ASCII	de	'm'	eh	53
0	codigo	ASCII	de	'n'	eh	54
0	codigo	ASCII	de	'o'	eh	55
0	codigo	ASCII	de	'p'	eh	56

C:\CE\For\bin\Debug\For.exe

0	codigo	ASCII	de	'q'	eh	56
0	codigo	ASCII	de	'r'	eh	57
0	codigo	ASCII	de	's'	eh	58
0	codigo	ASCII	de	't'	eh	59
0	codigo	ASCII	de	'u'	eh	60
0	codigo	ASCII	de	'v'	eh	61
0	codigo	ASCII	de	'w'	eh	62
0	codigo	ASCII	de	'x'	eh	63
0	codigo	ASCII	de	'y'	eh	64
0	codigo	ASCII	de	'z'	eh	65
0	codigo	ASCII	de	'{'	eh	66
0	codigo	ASCII	de	' '	eh	67
0	codigo	ASCII	de	'}'	eh	68
0	codigo	ASCII	de	'~'	eh	69
0	codigo	ASCII	de	'_'	eh	70
0	codigo	ASCII	de	'@'	eh	71
0	codigo	ASCII	de	'A'	eh	72
0	codigo	ASCII	de	'B'	eh	73
0	codigo	ASCII	de	'C'	eh	74
0	codigo	ASCII	de	'D'	eh	75
0	codigo	ASCII	de	'E'	eh	76
0	codigo	ASCII	de	'F'	eh	77
0	codigo	ASCII	de	'G'	eh	78
0	codigo	ASCII	de	'H'	eh	79
0	codigo	ASCII	de	'I'	eh	80

C:\CE\For\bin\Debug\For.exe

0	codigo	ASCII	de	'J'	eh	232
0	codigo	ASCII	de	'K'	eh	233
0	codigo	ASCII	de	'L'	eh	234
0	codigo	ASCII	de	'M'	eh	235
0	codigo	ASCII	de	'N'	eh	236
0	codigo	ASCII	de	'O'	eh	237
0	codigo	ASCII	de	'P'	eh	238
0	codigo	ASCII	de	'Q'	eh	239
0	codigo	ASCII	de	'R'	eh	240
0	codigo	ASCII	de	'S'	eh	241
0	codigo	ASCII	de	'T'	eh	242
0	codigo	ASCII	de	'U'	eh	243
0	codigo	ASCII	de	'V'	eh	244
0	codigo	ASCII	de	'W'	eh	245
0	codigo	ASCII	de	'X'	eh	246
0	codigo	ASCII	de	'Y'	eh	247
0	codigo	ASCII	de	'Z'	eh	248
0	codigo	ASCII	de	'['	eh	249
0	codigo	ASCII	de	']'	eh	250
0	codigo	ASCII	de	'^'	eh	251
0	codigo	ASCII	de	'_'	eh	252
0	codigo	ASCII	de	'`'	eh	253
0	codigo	ASCII	de	'a'	eh	254
0	codigo	ASCII	de	'b'	eh	255



Estruturas de Repetição: **break**

- O **break** permite encerrar um laço no meio de sua execução. Pode ser utilizado com
 - **while**, **do-while**, ou **for**
- A sintaxe é a seguinte:

```
while (expressaoBooleana1) {  
    comandos1;  
    if (expressaoBooleana2) break;  
    comandos2;  
}
```

```
do {  
    comandos1;  
    if (expressaoBooleana2) break;  
    comandos2;  
} while (expressaoBooleana1);
```

```
for (inicializacao; expressaoBooleana1; incrementos) {  
    comandos1; //...  
    if (expressaoBooleana2) break;  
    comandos2; //...  
}
```

- Dificulta a compreensão do código, pois há mais de uma condição de término a ser verificada, mas é útil quando precisamos encerrar o laço **no meio do bloco**, evitando que alguns comandos sejam executados.



Estruturas de Repetição: **break**

- Exemplo:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float num, media, soma=0;
6      int i, n;
7      printf("Digite o maximo de entradas: ");
8      scanf("%d", &n);
9
10     for(i=1; i<=n; i++) {
11         printf("Entre o valor %d (-1 para finalizar): ", i);
12         scanf("%f", &num);
13         if(num<0.0) break; //termina o laço
14         soma=soma+num;
15     }
16     media=soma/(i-1);
17     printf("Media = %f\n", media);
18 }
```

```
C:\CE\Break\bin\Debug\Break.exe
Digite o maximo de entradas: 10
Entre o valor 1 (-1 para finalizar): 2.0
Entre o valor 2 (-1 para finalizar): 4.0
Entre o valor 3 (-1 para finalizar): 6.0
Entre o valor 4 (-1 para finalizar): -1
Media = 4.000000
```



Estruturas de Repetição: **continue**

- O **continue** permite passar para a próxima repetição no meio do laço.
 - Pode ser utilizado no **while**, **do-while**, ou no **for**.
- A sintaxe é a seguinte:

```
while (expressaoBooleana1) {  
    comandos1;  
    if (expressaoBooleana2) continue;  
    comandos2;  
}
```

```
do {  
    comandos1;  
    if (expressaoBooleana2) continue;  
    comandos2;  
} while (expressaoBooleana1);
```

```
for (inicializacao; expressaoBooleana; incremento) {  
    comandos1; // ...  
    if (expressaoBooleana2) continue;  
    comandos2; // ...  
}
```

- Dificulta a compreensão do código, pois há mais de uma condição a ser verificada, mas é útil quando precisamos pular os próximos comandos **no meio do bloco**, indo direto para a próxima repetição.



Estruturas de Repetição: **continue**

- Exemplo: calcular o produto apenas dos números digitados que forem diferentes de zero:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i, num, produto;
6      for(i=1, produto=1; i<=5; i++) {
7          printf("Entre o valor %d:", i);
8          scanf("%d", &num);
9          //se num==0 pula os proximos comandos
10         if(num==0) continue;
11         produto*=num;
12         printf("Produto parcial: %d\n", produto);
13     }
14     printf("Produto final: %d\n", produto);
15 }
```

```
C:\CE\Continue\bin\Debug\Continue.exe
Entre o valor 1:2
Produto parcial: 2
Entre o valor 2:0
Entre o valor 3:3
Produto parcial: 6
Entre o valor 4:5
Produto parcial: 30
Entre o valor 5:2
Produto parcial: 60
Produto final: 60
```



Estruturas de Repetição:

Laços Aninhados

- Todos os laços vistos podem ser aninhados, uns dentro dos outros. Exemplos:

```
while (expressaoBooleana1) {  
    comandos1;  
    while (expressaoBooleana2) {  
        comandos2;  
        comandos3;  
    }  
    comandos4;  
}
```

```
while (expressaoBooleana1) {  
    comandos1;  
    do {  
        comandos2;  
        comandos3;  
    } while (expressaoBooleana1);  
    comandos4;  
}
```

```
for (inicializacao; expressaoBooleana1; incrementos) {  
    comandos1;  
    while (expressaoBooleana2) {  
        comandos2;  
        comandos3;  
    }  
    comandos4;  
}
```



Estruturas de Repetição: Laços Aninhados

- Exemplo:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i, j;
6      int total=0;
7
8      for (i=0; i<2; i++) {
9          for (j=0; j<3; j++) {
10             total = total + 1;
11             printf("i: %d j: %d total: %d\n", i, j, total);
12         }
13     }
14     printf("O valor final de total e: %d\n", total);
15 }
```

```
C:\CE\LacosAninhados\bin\Debug\La
i: 0 j: 0 total: 1
i: 0 j: 1 total: 2
i: 0 j: 2 total: 3
i: 1 j: 0 total: 4
i: 1 j: 1 total: 5
i: 1 j: 2 total: 6
O valor final de total e: 6
```

- Qual o valor final de **total**, impresso pelo programa?



Estruturas de Repetição:

Laços Aninhados

- Exemplo: impedir entradas inválidas, para calcular o produto apenas dos números digitados que forem diferentes de zero:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i, num, produto;
6      for (i=1, produto=1; i<=5; i++) {
7          //Le a entrada e prossegue se não for zero:
8          do {
9              printf("Entre o valor %d:", i);
10             scanf("%d", &num);
11             if (num==0) printf("O valor nao pode ser zero!\n");
12         } while (num==0);
13         //Calcula o produto:
14         produto*=num;
15         printf("Produto parcial: %d\n", produto);
16     }
17     printf("Produto final: %d\n", produto);
18 }
```

```
C:\CE\Continue\bin\Debug\Continue
Entre o valor 1:2
Produto parcial: 2
Entre o valor 2:0
O valor nao pode ser zero!
Entre o valor 2:0
O valor nao pode ser zero!
Entre o valor 2:1
Produto parcial: 2
Entre o valor 3:3
Produto parcial: 6
Entre o valor 4:4
Produto parcial: 24
Entre o valor 5:1
Produto parcial: 24
Produto final: 24
```



Estruturas de Repetição: resumo

- Como escolher entre as estruturas de repetição?

```
if (numero de vezes é conhecido) {  
    usa-se o for  
}  
else {  
    if (teste precisa ser feito no início) {  
        usa-se while  
    } else {  
        usa-se do-while  
    }  
}
```

- Porém, na maioria dos casos, pode-se escolher mais de uma opção para resolver cada tipo de problema e acaba sendo uma questão de preferência pessoal.



Exemplo de Cálculo de Maior ou Máximo

```
#include <stdio.h>

int main() {

    //entrada
    int num_medalhas = 0;
    int paises = 100;

    //saida
    int max_medalhas = 0;

    int i = 1;
    for(; i <= paises; i++){

        printf("Digite o numero de medalhas do pais %d:", i);
        scanf("%d", &num_medalhas);
        if(num_medalhas > max_medalhas){
            max_medalhas = num_medalhas;
        }
    }

    printf("O maior numero de medalhas foi: %d", max_medalhas);

    return 0;
}
```



Exemplo de Cálculo de Média

```
#include <stdio.h>

int main(){

    //entrada
    int nota = 0;
    int num_notas = 5;

    //saida
    float media = 0;

    int soma_notas = 0;

    int i = 1;
    for(;i<=num_notas;i++){
        printf("Digite a nota do arbitro %d:",i);
        scanf("%d",&nota);
        soma_notas = soma_notas + nota;
    }

    media = (float)soma_notas/num_notas;
    printf("O media das notas foi %.2f",media);

    return 0;
}
```



Atividade 1

1. Escreva um algoritmo que lê 50 números inteiros e em seguida mostra a soma de todos os ímpares lidos.
2. Altere o algoritmo anterior para que ele considere apenas a soma dos ímpares que estejam entre 100 e 200.
3. Construa um algoritmo que leia um conjunto de 20 números inteiros e mostre qual foi o maior e o menor valor fornecido.
4. Altere o programa anterior para que ele não permita a entrada de valores negativos.



Atividade 1.1

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

    int num;
    int soma = 0;

    for (i=0;i<50;i++) {
        printf("Digite o numero: ");
        scanf("%d",&num);
        if (num%2==1){
            soma = soma + num;
        }
    }
    printf("\nA soma eh: %d",soma);
    return 0;
}
```



Atividade 2

- Faça um programa para imprimir a tabuada de 1 a 9 utilizando laços de repetição. A saída do programa deve ser do tipo:

```
1x1 = 1
1x2 = 2
...
1x9 = 9
//linha em branco
2x1 = 2
2x2 = 4
...
2x9 = 18
//linha em branco
3x1 = 3
3x2 = 6
...
...
9x1 = 9
...
9x9 = 81
```



Atividade 3

- Em uma disciplina com três exercícios, um professor deseja saber qual a maior e a menor nota de cada exercício e qual a maior e a menor média final. Faça um programa para auxiliá-lo nesta tarefa.
 1. O programa deve receber como entrada três notas de cada aluno em uma linha;
 2. Deve perguntar ao usuário se deseja inserir as notas de outro aluno, e:
 - Caso a resposta seja “sim” deve solicitar os dados do próximo aluno.
 - Caso a resposta seja “não” deve mostrar a maior e a menor nota do primeiro, do segundo, e do terceiro exercício e a maior e a menor média.

– Obs.: As notas devem ser impressas com duas casas decimais, de acordo com o exemplo abaixo:

```
Entre com as notas do 1° aluno: 5.0 6.0 7.0
Deseja inserir as notas de outro aluno? (s/n):s
Entre com as notas do 2° aluno: 4.0 8.0 5.0
Deseja inserir as notas de outro aluno? (s/n):s
Entre com as notas do 3° aluno: 5.0 6.0 9.0
Deseja inserir as notas de outro aluno? (s/n):n

| 1°Exercicio | 2°Exercicio | 3°Exercicio |      Media
|             |             |             |
| Maior|Menor | Maior|Menor | Maior|Menor | Maior|Menor
```



Atividade 4

- Fazer um programa para calcular a raiz quadrada de um número positivo usando o roteiro abaixo (baseado no método de aproximações sucessivas de Newton-Raphson).
- Seja Y o número para o qual se quer computar a raiz quadrada:
 - A primeira aproximação para a raiz quadrada de Y é dada por:

$$X_1 = \frac{Y}{2}$$

- As demais aproximações são calculadas pela seguinte fórmula de recorrência:

$$X_{i+1} = X_i - \frac{X_i^2 - Y}{2X_i}, \text{ onde } X_i \text{ é a aproximação anterior e } X_{i+1} \text{ a nova.}$$

- A aproximação deve continuar até que o valor obtido mude pouco com o passar das iterações (menos de $e = 0.1$), Ou seja:

$$|X_{i+1} - X_i| < e, \text{ onde } e = 0.1$$

- Compare o resultado obtido com o resultado produzido pela função **sqrt()** disponível na biblioteca `<math.h>`:

$$|X_n - \text{sqrt}(Y)|, \text{ onde } n \text{ é o número de iterações realizadas}$$

- Obs.: O valor absoluto $| \cdot |$ pode ser calculado através da função **fabs()**, disponível na biblioteca `<math.h>`.



Atividade 5: 1EE – 2014.1

1º quesito: Um revista especializa deseja calcular estatísticas sobre a primeira fase da copa do mundo do Brasil. Com o objetivo de automatizar este cálculo, você deverá criar um programa na linguagem C para ajudar os repórteres da publicação. A primeira fase da competição possui 34 times. O programa deverá ler os dados dos 34 times: 1) Número de vitórias; 2) Número de derrotas; 3) Número de empates; 4) Número de gols efetuados; 5) Número de gols sofridos; 6) Quantidade de cartões vermelhos recebidos (Expulsões).

O programa deverá exibir na tela:

- a) O **maior** número de vitórias;
- b) O **maior** número de derrotas;
- c) O saldo **médio** de gols geral (o saldo de gols de um time = quantidade de gols efetuados – quantidade de gols recebidos);
- d) Percentagem de times **com mais de** 3 cartões vermelhos.

Obs.: É proibido o uso de comando goto;

Os valores devem ser impressos na tela com precisão de **duas casas decimais** depois da virgula.



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

```
int main() {
```

```
    int vitorias, derrotas, empates, gols_efe, gols_sof, cartoes, saldo_gols;
```

```
    int m_vitorias=0, m_derrotas=0, soma_saldo_gols=0, cont_cartoes_vermelhos=0;
```

```
    float media_gols;
```

```
    int times = 34, i;
```

```
    for (i=0;i<times;i++) {
```

```
        printf("TIME %d\n",i);
```

```
        printf("Digite o numero de vitorias: ");
```

```
        scanf("%d",&vitorias);
```

```
        printf("Digite o numero de derrotas: ");
```

```
        scanf("%d",&derrotas);
```

```
        printf("Digite o numero de empates: ");
```

```
        scanf("%d",&empates);
```

```
        printf("Digite o numero de gols efetuados: ");
```

```
        scanf("%d",&gols_efe);
```

```
        printf("Digite o numero de gols sofridos: ");
```

```
        scanf("%d",&gols_sof);
```

```
        printf("Digite o numero de cartoes vermelhos: ");
```

```
        scanf("%d",&cartoes);|
```



```
    if (vitorias>m_vitorias) {
        m_vitorias = vitorias;
    }

    if (derrotas > m_derrotas) {
        m_derrotas = derrotas;
    }

    saldo_gols = gols_efe - gols_sof;
    soma_saldo_gols += saldo_gols;

    if (cartoes > 3) {
        cont_cartoes_vermelhos++;
    }
}

printf("\nMaior numero de vitorias: %d",m_vitorias);
printf("\nMaior numero de derrotas: %d",m_derrotas);
printf("\nSaldo medio do gols: %.2f",(float)soma_saldo_gols/times);
printf("\nPercentual de times com mais de 3 cartoes vermelhos:%.2f",
(float)cont_cartoes_vermelhos/times);

return 0;
```