

Estruturas de Repetição

Rafael Beserra Gomes

UFRN

Material compilado em 13 de dezembro de 2017.

Licença desta apresentação:



<http://creativecommons.org/licenses/>

Estruturas de repetição

- Em determinados momentos queremos **repetir** uma mesma instrução ou um bloco de instruções por várias vezes
- Por exemplo: escrever na tela os números de 1 a 100

```
1 #include <stdio.h>
2
3 int main() {
4
5     printf("1\n");
6     printf("2\n");
7     printf("3\n");
8     printf("4\n");
9     printf("5\n");
10    printf("6\n");
11    printf("7\n");
12
13    return 0;
14 }
```

- Por que é inviável escrever o mesmo bloco de instruções diversas vezes?

■ Outro exemplo: quais os divisores de um número?

```
1 #include <stdio.h>
2
3 int main() {
4
5     int n;
6
7     printf("Digite um numero: ");
8     scanf("%d", &n);
9
10    if(n%1 == 0)
11        printf("1 divide %d\n", n);
12    if(n%2 == 0)
13        printf("2 divide %d\n", n);
14    if(n%3 == 0)
15        printf("3 divide %d\n", n);
16
17    return 0;
18 }
```

- Você pode não saber quantas vezes repetir, pois o número de repetições pode ser em função de alguma variável

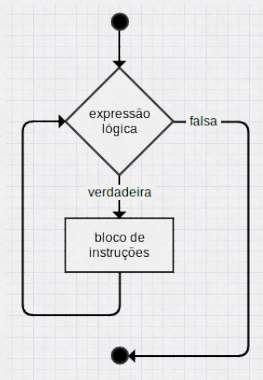
- Uma **estrutura de repetição** permite repetir um **bloco de instruções** por nenhuma ou mais vezes
- Cada repetição é chamada de **iteração**
- O **bloco de instruções** pode conter qualquer instrução válida que vimos até agora, incluindo condicionais e outras estruturas de repetição (estruturas de repetição aninhada)

Veremos as seguintes estruturas de repetição:

- while
- do/while
- for

Estrutura de repetição **while** (enquanto)

```
while(<expressão lógica>) {  
  <instrução 1>  
  <instrução 2>  
  <...>  
  <instrução n>  
}
```



- o bloco de instruções é executado **enquanto** o valor da expressão lógica **for verdadeiro**
- bastante útil quando não sabemos o número de iterações

Exemplo: escrevendo os números de 1 a 100

```
1 #include <stdio.h>
2
3 int main() {
4
5     int i;
6
7     i = 1;
8     while(i <= 100) {
9         printf("%d\n", i);
10        i = i + 1;
11    }
12
13    return 0;
14 }
```


Exemplo: quais os divisores de um número?

```
1 #include <stdio.h>
2
3 int main() {
4
5     int i, n;
6     printf("Digite um numero: ");
7     scanf("%d", &n);
8
9     i = 1;
10    while(i <= 100) {
11        if(n%i == 0) {
12            printf("%d\n", i);
13        }
14        i = i + 1;
15    }
16
17    return 0;
18 }
```



Cuidado com a **indentação**!!

```
1 #include <stdio.h>
2
3 int main() {
4
5     //nao indente dessa forma!
6     int i, n;
7     printf("Digite um numero: ");
8     scanf("%d", &n);
9
10    i = 1;
11    while(i <= 100) {
12        if(n%i == 0) {
13            printf("%d\n", i);
14        }
15        i = i + 1;
16    }
17
18    return 0;
19 }
```

Exercício em sala

Escreva um programa em C que leia um número inteiro **n**. Depois o programa deve escrever na tela todos os números pares de 1 a **n** e depois todos os números ímpares de 1 a **n**.

Exemplo: Digite n: **8**
Pares: 2 4 6 8
Ímpares: 1 3 5 7

Exercício em sala

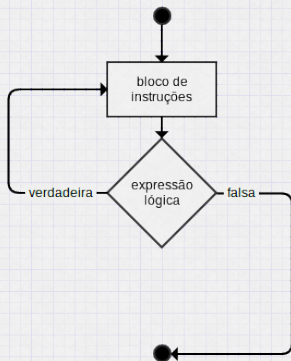
Problema $3*n + 1$: dado um número n , se este número for par, divida-o por 2 e se for ímpar, multiplique por 3 e some 1. Repita o processo até chegar no número 1.

Escreva um programa em C que leia um número inteiro n e escreva na tela a sequência gerada pelas regras acima.

Exemplo: Digite n: 5
5 16 8 4 2 1

Estrutura de repetição **do/while** (faça/enquanto)

```
do {  
  _<instrução 1>  
  _<instrução 2>  
  _<...>  
  _<instrução n>  
} while(<expressão lógica>);
```



- o bloco de instruções é executado **e continua sendo executado enquanto** o valor da expressão lógica **for verdadeiro**
- use-o quando o bloco de instruções precede o primeiro teste

Exemplo: lê dois números a e b. O valor de b deve ser solicitado novamente enquanto for 0. Em seguida, escreve o resultado da divisão $\frac{a}{b}$.

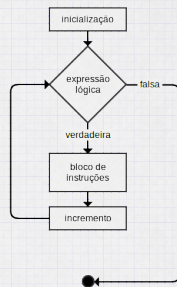
```
1 #include <stdio.h>
2
3 int main() {
4
5     int a, b;
6
7     printf("Digite um numero inteiro A: ");
8     scanf("%d", &a);
9     do {
10         printf("Digite um numero inteiro B: ");
11         scanf("%d", &b);
12     } while(b == 0);
13
14     printf("A dividido por B = %f\n", ((float)a)/b);
15     return 0;
16 }
```

Exercício em sala

Acerte a senha: escreva um programa que leia um número inteiro. O programa deve prosseguir somente quando esse número for igual a uma senha que você definir no código. Escreva em seguida o número de tentativas para acertar a senha.

Estrutura de repetição **for** (para)

```
for(<inicialização>; <expressão lógica>; <incremento>)  
{  
  _<instrução 1>  
  _<instrução 2>  
  _<...>  
  _<instrução n>  
}
```



- **inicialização**: executado antes da estrutura de repetição
- **expressão lógica**: avaliado antes de cada iteração (encerra caso falso)
- **incremento**: executado após cada iteração

Exemplo: escrevendo os números de 1 a 100

```
1 #include <stdio.h>
2
3 int main() {
4
5     int i;
6
7     for(i = 1; i <= 100; i++) {
8         printf("%d\n", i);
9     }
10
11     return 0;
12 }
```

Exemplo: quais os divisores de um número?

```
1 #include <stdio.h>
2
3 int main() {
4
5     int i, n;
6     printf("Digite um numero: ");
7     scanf("%d", &n);
8     for(i = 1; i <= n; i++) {
9         if(n%i == 0) {
10             printf("%d\n", i);
11         }
12     }
13     return 0;
14 }
```

Exercício em sala

Escreva um programa em C que leia um número inteiro **n** e escreva na tela se esse número é primo.

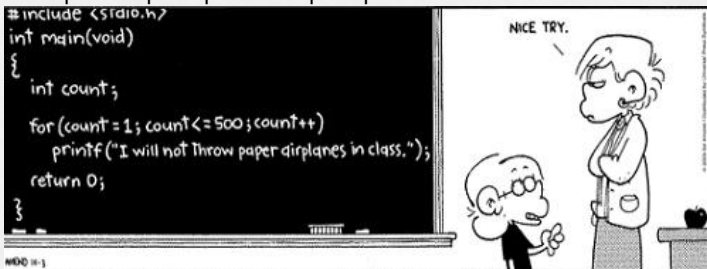
Escolhendo entre for e while

- **reflita** como o uso de uma ou outra estrutura de repetição pode afetar a legibilidade do código

```
1 #include <stdio.h>
2
3 int main() {
4
5     int a, b;
6
7     scanf("%d", &a);
8     //aqui eh melhor usar um do/while como fizemos anteriormente
9     for(b = 0; b == 0;) {
10         scanf("%d", &b);
11     }
12     printf("%f\n", ((float)a/b));
13
14     return 0;
15 }
```

- no geral:

- **for**: aplicada principalmente para percorrer um intervalo numérico



- **while**: aplicada principalmente quando o número de iterações é desconhecido

Estrutura de repetição: break e continue

■ *break*: encerra imediatamente a estrutura de repetição

```
1 #include <stdio.h>
2 #include <unistd.h>
3 int main() {
4     int n, i, soma = 0;
5     printf("Soma 10 numeros (mas nao entre com zero!)\n");
6     for(i = 1; i <= 10; i++) {
7         printf("Digite um numero: ");
8         scanf("%d", &n);
9         if(n == 0) {
10             printf("Avissei! Apagando o seu HD (please wait)\n");
11             sleep(3);
12             printf("Brincadeira!\n");
13             break;
14         }
15         soma = soma + n;
16     }
17     printf("Soma = %d\n", soma);
18     return 0;
19 }
```

Exercício em sala



Escreva um programa que leia um número n e escreva na tela se o número é primo ou não. Ao encontrar um divisor diferente de 1 e n , não realize mais outros testes (use `break`) pois já concluímos que n não é primo.

- **continue**: encerra imediatamente a iteração atual¹ e parte para a próxima iteração


```
1 #include <stdio.h>
2 #include <unistd.h>
3 int main() {
4     int n, i, soma = 0;
5     printf("Soma 10 numeros (mas nao entre com negativo!)\n");
6     for(i = 1; i <= 10; i++) {
7         printf("Digite um numero: ");
8         scanf("%d", &n);
9         if(n < 0) {
10             printf("Voce sera punido com 5s de espera\n");
11             sleep(5);
12             continue;
13         }
14         soma = soma + n;
15     }
16     printf("Soma = %d\n", soma);
17     return 0;
18 }
```

¹ainda realiza o incremento e o teste no caso do for

Notas adicionais

-  Um **loop infinito** ocorre quando a estrutura de repetição não encerra. Nesses casos, para encerrar o programa aperte *ctrl+c*
-  Pode omitir qualquer uma das 3 expressões do for:

```
1 for(;;) {  
2     printf("Loop infinito\n");  
3 }
```

-  Pode usar várias operações na **inicialização** e no **incremento**

```
1 for(i = 0, j = 1; i <= 10; i++, j *= 2) {  
2     printf("2 elevado a %d = %d\n", i, j);  
3 }
```