

# Comandos Repetitivos

Alexandre Mello

Fatec Campinas

2018

# Roteiro

- 1 Comandos Repetitivos
- 2 Comando **while**
- 3 Comando **do-while**
- 4 O comando **for**
- 5 Exemplos com Laços
  - Variável acumuladora : Soma de números
  - Variável acumuladora: Calculando Potências de 2
  - Variável acumuladora: Calculando o valor de  $n!$
- 6 Comandos **continue** e **break**
- 7 Exercícios

# Comandos Repetitivos

- Até agora vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação a executar ou não um bloco de comandos.
- Entretanto, eventualmente é necessário executar um bloco de comandos várias vezes para se obter o resultado esperado.

# Introdução

- Ex.: Programa que imprime todos os números de 1 até 4.
- Será que dá pra fazer com o que já sabemos?

```
printf("1");  
printf("2");  
printf("3");  
printf("4");
```

# Introdução

- Ex.: Programa que imprime todos os números de 1 até 100.

```
printf("1");  
printf("2");  
printf("3");  
printf("4");  
/*repete 95 vezes a linha acima*/  
printf("100");
```

# Introdução

- Ex.: Programa que imprime todos os números de 1 até  $n$  (informado pelo usuário).

```
printf("1");  
if (n>=2)  
    printf("2");  
if (n>=3)  
    printf("3");  
/*repete 96 vezes o bloco acima*/  
if (n>=100)  
    printf("100");
```

- Note que o programa é válido para  $n \leq 100$ .

# Comando **while**

- Estrutura:

```
while ( condição )  
    comando;
```

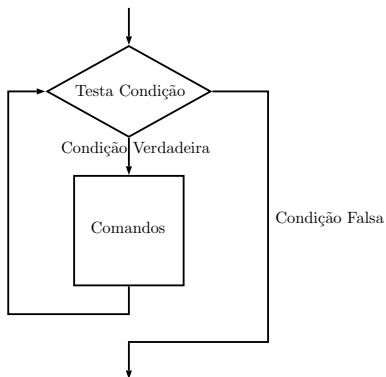
- Ou:

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

- Enquanto a condição for verdadeira ( $\neq 0$ ), ele executa o(s) comando(s).

# Comando **while**

- Passo 1: Testa a condição. Se a condição for verdadeira vai para o Passo 2.
- Passo 2.1: Executa os comandos.
- Passo 2.2: Volta para o Passo 1.





# Comando **while**

Imprimindo os 100 primeiros números inteiros:

```
int i=1;
while (i<=100)
{
    printf("%d ",i);
    i++;
}
```

# Comando **while**

Imprimindo os  $n$  primeiros números inteiros:

```
int i=1,n;  
scanf("%d",&n);  
while (i<=n)  
{  
    printf("%d ",i);  
    i++;  
}
```

# Comando **while**

- 1. O que acontece se a condição for falsa na primeira vez?

```
while (a!=a)
    a=a+1;
```

- 2. O que acontece se a condição for sempre verdadeira?

```
while (a == a)
    a=a+1;
```

# Comando **while**

- 1. O que acontece se a condição for falsa na primeira vez?

```
while (a!=a)
    a=a+1;
```

**Resposta:** Ele nunca entra na repetição (no laço).

- 2. O que acontece se a condição for sempre verdadeira?

```
while (a == a)
    a=a+1;
```

**Resposta:** Ele entra na repetição e nunca sai (laço infinito).

# Comando **do-while**

- Estrutura:

```
do
    comando;
while ( condição );
```

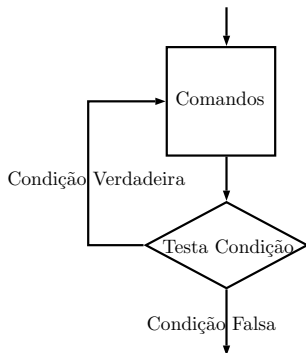
- Ou:

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

- Diferença do **while**: sempre executa comandos na primeira vez. Teste condicional é feito por último.

# Comando **do-while**

- Passo 1: Executa comandos.
- Passo 2: Testa condição. Se for verdadeira vai para Passo 1.



# Comando **do-while**

Imprimindo os 100 primeiros números inteiros:

```
int i;  
i=1;  
do{  
    printf("\n %d",i);  
    i = i+1;  
}while(i<= 100);
```

# Comando **do-while**

Imprimindo os  $n$  primeiros números inteiros:

```
int i, n;  
i=1;  
scanf("%d",&n);  
do{  
    printf("\n %d",i);  
    i++;  
}while(i<=n);
```

- O que acontece se o usuário digitar 0?
- O que acontece se usarmos o comando **while**?



# O comando **for**

- Estrutura:

```
for (início ; condição ; passo)
    comando do bloco;
```

- Ou:

```
for (início ; condição ; passo) {
    comandos do bloco;
}
```

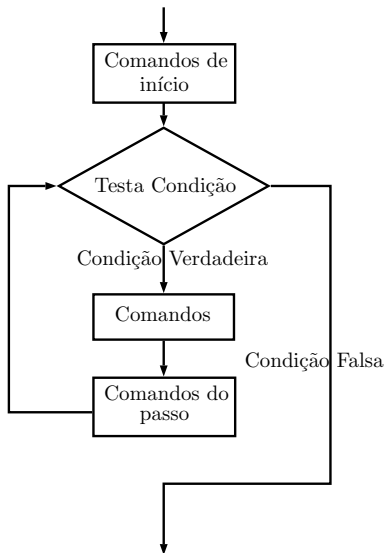
- **Início:** Uma ou mais atribuições, separadas por “,”.
- **Condição:** Comandos são executados enquanto a condição for verdadeira.
- **Passo:** Um ou mais comandos separados por “,”. Os comandos do passo são executados após os comandos do bloco.

# O comando **for**

```
for (início ; condição ; passo) {  
    comandos do bloco;  
}
```

- Passo 1: Executa comandos em "início".
- Passo 2: Testa condição. Se for verdadeira vai para passo 3.
- Passo 3.1: Executa comandos do bloco.
- Passo 3.2: Executa comandos em "passo".
- Passo 3.2: Volta ao Passo 2.

# O comando **for**



# O Comando **for**

O **for** é equivalente a seguinte construção utilizando o **while**:

```
início;  
while(condição){  
    comandos;  
    passo;  
}
```

# O Comando **for**

Imprimindo os 100 primeiros números inteiros:

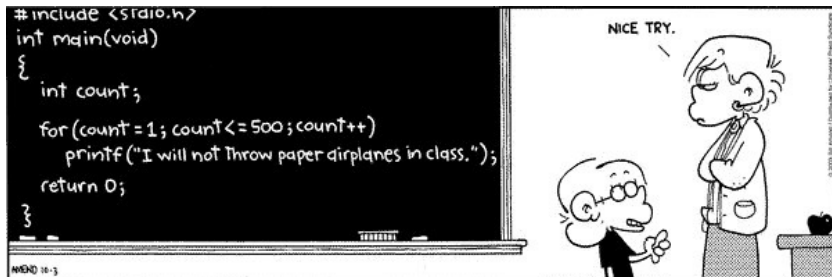
```
int i;  
for(i=1; i<= 100; i=i+1){  
    printf("\n %d",i);  
}
```

# O Comando **for**

Imprimindo os  $n$  primeiros números inteiros:

```
int i, n;  
scanf("%d",&n);  
for(i=1; i<=n; i++){  
    printf("\n %d",i);  
}
```

# I'll not throw paper airplanes in class



# Variável Acumuladora

- Vamos ver alguns exemplos de problemas que são resolvidos utilizando laços.
- Há alguns padrões de solução que são bem conhecidos, e são úteis em diversas situações.
- O primeiro padrão deles é o uso de uma “variável acumuladora”.

## Problema

Ler um inteiro positivo  $n$ , em seguida ler  $n$  números do teclado e apresentar a soma destes.



# Soma de números

- Como  $n$  não é definido a priori, não podemos criar  $n$  variáveis e depois somá-las.
- A idéia é criar uma variável acumuladora que a cada iteração de um laço acumula a soma de todos os números lidos até então.

```
acumuladora = 0 // inicialmente não somamos nada
```

```
Repita n vezes
```

```
    Leia um número aux
```

```
    acumuladora = acumuladora + aux
```

# Soma de números

- Podemos usar qualquer estrutura de laço de C para esta solução.
- Abaixo temos uma solução utilizando o comando **for**.

```
acu = 0;
for(i=1; i<=n; i++){
    printf("Digite um numero: ");
    scanf("%d", &aux);
    acu = acu + aux;
}
```

# Soma de números

Código completo:

```
#include <stdio.h>

int main(){
    int acu, i, n, aux;

    printf("Entre com n: ");
    scanf("%d", &n);
    acu = 0;
    for(i=1; i<=n; i++){
        printf("Digite um numero: ");
        scanf("%d", &aux);
        acu = acu + aux;
    }
    printf("Soma total: %d\n", acu);
}
```

# Calculando potências de 2

Mais um exemplo:

## Problema

Leia um inteiro positivo  $n$ , e imprima as potências:  $2^0, 2^1, \dots, 2^n$ .

# Calculando potências de 2

- Usamos uma variável acumuladora que no início da  $i$ -ésima iteração de um laço, possui o valor  $2^i$ .
- Imprimimos este valor e atualizamos a acumuladora para a próxima iteração, multiplicando esta variável por 2.

acumuladora = 1 // Corresponde a  $2^0$

Para  $i=0$  até  $n$  faça:

    imprima acumuladora

    acumuladora = acumuladora \* 2

# Calculando potências de 2

- A solução pode ser obtida utilizando-se o laço **for**.

```
pot = 1; // corresponde a 2^0
for(i=0; i<=n; i++){
    printf("%d\n", pot);
    pot = pot * 2;
}
```

- Mas vamos fazer este exemplo utilizando o comando **while**.

# Calculando Potências de 2

Em C:

```
int i, n, pot;

scanf("%d",&n);
pot = 1;
i = 0;
while(i <= n){
    printf("2^%d = %d\n",i,pot);
    pot = pot *2;
    i++;
}
```

# Calculando Potências de 2

Programa completo:

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

```
int main(){  
    int i, n, pot;  
  
    scanf("%d",&n);  
    pot = 1;  
    i = 0;  
    while(i <= n){  
        printf("2^%d = %d\n",i,pot);  
        pot = pot *2;  
        i++;  
    }  
}
```



# Calculando o valor de $n!$

## Problema

Fazer um programa que lê um valor inteiro positivo  $n$  e calcula o valor de  $n!$ .

- Lembre-se que  $n! = n * (n - 1) * (n - 2) * \dots * 2 * 1$ .

# Calculando o valor de $n!$

- Criamos uma variável acumuladora que no início da  $i$ -ésima iteração de um laço armazena o valor de  $(i - 1)!$ .
- Durante a  $i$ -ésima iteração atualizamos a variável acumuladora multiplicando esta por  $i$  obtendo  $i!$ .
- No fim do laço, após  $n$  iterações, teremos na acumuladora o valor de  $n!$ .

```
acumuladora = 1 //corresponde a 0!
```

```
Para i=1 até n faça:
```

```
    acumuladora = acumuladora * i
```

```
    i = i + 1
```

# Calculando o valor de $n!$

Em C:

```
int i, n, acu;  
  
scanf("%d", &n);  
acu = 1;  
  
for(i=1; i <= n; i++){  
    acu = acu * i;  
}
```

# Calculando o valor de $n!$

Programa completo:

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

```
int main(){
```

```
    int i, n, acu;
```

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

```
    acu = 1;
```

```
    for(i=1; i <= n; i++){
```

```
        acu = acu * i;
```

```
    }
```

```
    printf("Fatorial de %d e: %d\n",n, acu);
```

```
}
```

# Laços e o comando **break**

- O comando **break** faz com que a execução de um laço seja terminada, passando a execução para o próximo comando depois do final do laço.

```
int i;  
for(i = 1; i<= 10 ; i++){  
    if(i >= 5)  
        break;  
    printf("%d\n",i);  
}  
printf("Terminou o laço");
```

O que será impresso?

# Laços e o comando **break**

- O comando **break** faz com que a execução de um laço seja terminada, passando a execução para o próximo comando depois do final do laço.

```
int i;  
for(i = 1; i<= 10 ; i++){  
    if(i >= 5)  
        break;  
    printf("%d\n",i);  
}  
printf("Terminou o laço");
```

O que será impresso?

**Resposta:** Os números de 1 até 4 e depois a frase “Terminou o laço”.

# Laços e o comando **break**

- Assim como a “condição” em laços, o comando **break** é utilizado em situações de parada de um laço.

Exe.: Imprimindo os números de 1 até 10.

```
int i;
for(i = 1; ; i++){
    if(i > 10)
        break;
    printf("%d\n",i);
}
```

é equivalente a:

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

# Laços e o comando **continue**

- O comando **continue** faz com que a execução de um laço seja alterada para final do laço.

```
int i;  
for(i = 1; i<= 10 ; i++){  
    if(i == 5)  
        continue;  
    printf("%d\n",i);  
}  
printf("Terminou o laço");
```

O que será impresso?



# Laços e o comando **continue**

- O **continue** faz com que a execução de um laço seja alterada para final do laço.

```
int i;
for(i = 1; i<= 10 ; i++){
    if(i == 5)
        continue;
    printf("%d\n",i);
}
printf("Terminou o laço");
```

O que será impresso?

**Resposta:** Os números de 1 até 10, exceto o número 5, e depois a frase “Terminou o laço”.

# Laços e o comando **continue**

- O **continue** é utilizado em situações onde comandos dentro do laço só devem ser executados caso alguma condição seja satisfeita.

Exe.: Imprimindo área de um círculo, mas apenas se raio for par (e entre 1 e 10).

```
int r;
double area;
for(r = 1; r<= 10 ; r++){
    if( (r % 2) != 0) //se número for ímpar pulamos
        continue;
    area = 3.1415*r*r;
    printf("%lf\n",area);
}
```

Mas note que poderíamos escrever algo mais simples:

```
int r;
double area;
for(r = 2; r<= 10 ; r = r+2){
    area = 3.1415*r*r;
    printf("%lf\n",area);
}
```

# Exercício

- Faça um programa que imprima um menu de 4 pratos na tela e uma quinta opção para sair do programa. O programa deve imprimir o prato solicitado. O programa deve terminar quando for escolhido a quinta opção.

# Exercício

- Faça um programa que lê dois números inteiros positivos  $a$  e  $b$ . Utilizando laços, o seu programa deve calcular e imprimir o valor  $a^b$ .

# Exercício

- Faça um programa que lê um número  $n$  e que computa e imprima o valor

$$\sum_{i=1}^n i.$$

OBS: Não use fórmulas como a da soma de uma P.A.

# Exercício

- Faça um programa que lê um número  $n$  e imprima os valores entre 2 e  $n$  que são divisores de  $n$ .

# Exercício

- Faça um programa que lê um número  $n$  e imprima os valores

$$\sum_{i=1}^j i$$

para  $j$  de 1 até  $n$ , um valor por linha.