

Introdução à Programação



Estruturas de Repetição

Repetição de Comandos...

$$\sum_{i=0}^n i$$

```
int somatorio(int n) {  
    int soma = 0;  
    int i = 0;  
    soma = soma + i;  
    i++;  
    soma = soma + i;  
    i++;  
    soma = soma + i;  
    ...  
}
```

Faz esta seqüência de comandos **n** vezes

E haja *copy&paste!*

Tópicos da Aula

- ◆ Hoje, aprenderemos a usar estruturas de repetição para programas mais complexos
 - Necessidade de estruturas de repetição
 - Apresentação do conceito de laço (*loop*)
 - O comando *for*
 - O comando *while*
 - O comando *do-while*
 - Diferenças entre os comandos de repetição
- ◆ Veremos também, como podemos alterar o fluxo de laços
 - Comandos *break* e *continue*

Necessidade de Estruturas de Repetição

- ◆ Na resolução de problemas em programação, freqüentemente, precisamos repetir uma mesma seqüência de comandos várias vezes
- ◆ Na maioria dos casos, não sabemos de antemão quantas vezes a seqüência de comandos será repetida
- ◆ A linguagem de programação deve fornecer uma estrutura (comando) que permita a execução repetida de mesma seqüência de comandos
 - Evita esforço do programador
 - Permite que o programador não tenha que saber quantas vezes o comando será executado

Estruturas de Repetição

- ◆ Permite repetir diversas vezes um comando ou seqüência de comandos
 - Cada repetição de um comando ou seqüência de comandos é denominada de **iteração**
- ◆ São geralmente conhecidos como *loops*(laços)
- ◆ Da mesma forma que comandos condicionais, são controladas por expressões *booleanas*
- ◆ C oferece 3 tipos de estruturas(comandos) de repetição:
 - O laço *for*
 - O laço *while*
 - O laço *do-while*

O Comando `for`

```
for (i = 0; i < valor; i = i+1)  
    corpo
```

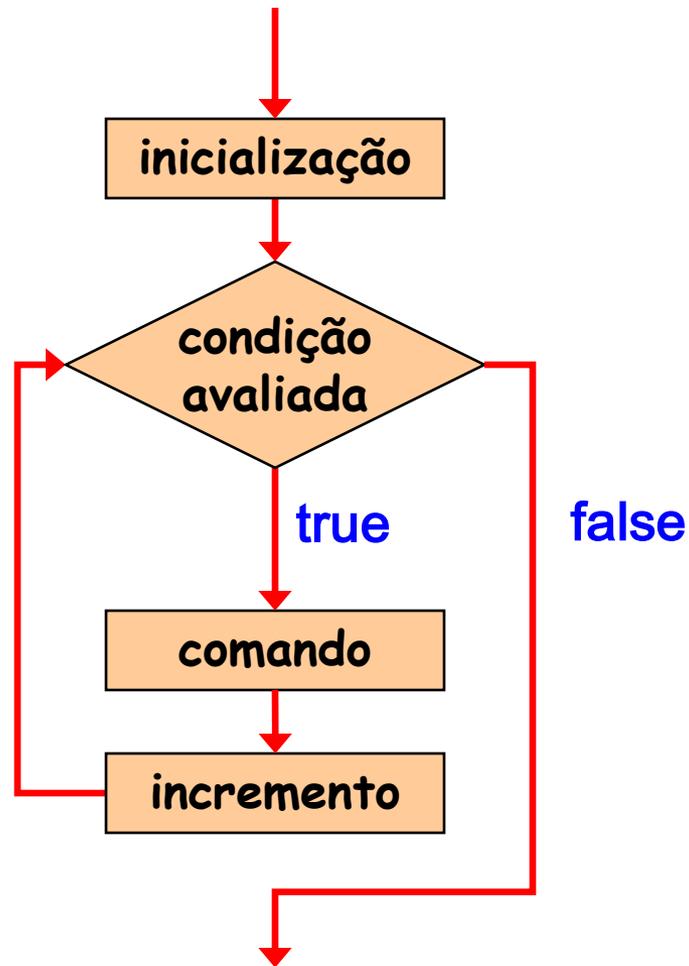
- ◆ Executa `corpo` um número específico de vezes: `valor` vezes
- ◆ Neste exemplo, na primeira execução de `corpo`, o valor de `i` é 0
- ◆ O valor de `i` é incrementado após cada execução de `corpo`
- ◆ Variável `i` **deve ser declarada** antes de se utilizar o comando `for`
 - **Variável de controle**

A Forma Geral do Comando `for`

`for` (inicialização ; condição ; incremento)
corpo

- ◆ inicialização e incremento podem ser praticamente quaisquer comandos
- ◆ condição pode ser qualquer expressão booleana
- ◆ inicialização geralmente inicializa a variável de controle do `for`
- ◆ incremento geralmente incrementa a variável `for`

Fluxo de Controle do Laço for



Examinando o Comando *for*

A *inicialização*
é executada uma só vez
antes do laço começar



O *comando* é
executado enquanto
condição for verdadeira



```
for ( inicialização ; condição ; incremento )  
    comando ;
```

O *incremento* é executado ao fim de
cada iteração



Cabeçalho do *for*

Entendendo o Comando for

```
int somatorio(int n) {  
    int valor;  
    int soma = 0;  
    for (valor = 0; valor <= n; valor++)  
        soma = soma + valor;  
    return soma;  
}
```

Variável
valor é
inicializada
com 0

Comando será
realizado
enquanto
valor for
menor ou
igual a **n**

A cada
iteração,
valor é
incrementado
em 1

Entendendo o Comando for

```
int somatorio(int n) {  
    int valor;  
    int soma = 0;  
    for (valor = 0; valor <= n; valor++)  
        soma = soma + valor;  
    return soma;  
}
```

Se **n** for
menor do que
0, não
se executa o
corpo do **for**

É executado
depois do
for

Entendendo o Comando for

```
int somatorio(int n) {  
    int soma = 0;  
    int valor;  
    for (valor = 0; valor <= n; valor++) {  
        soma = soma + valor;  
        printf("Soma Parcial:%d\n", soma);  
    }  
    printf("Soma Total:%d", soma);  
    return soma;  
}
```

Corpo do **for**
pode ser
composto por
bloco de
comandos

Modificando o Incremento do for

```
int somatorio(int n) {  
    int soma = 0;  
    int valor;  
    for (valor = n; valor >= 0; valor--){  
        soma = soma + valor;  
        printf("Soma Parcial:%d\n", soma) ;  
    }  
    printf("Soma Total:%d", soma) ;  
    return soma;  
}
```

valor agora é
decrementado

Modificando o Incremento do for

```
int somatorioPares(int n) {  
    int soma = 0;  
    int valor;  
    for (valor = 0; valor <= n; valor = valor + 2) {  
        soma = soma + valor;  
        printf("Soma Parcial:%d\n", soma);  
    }  
    printf("Soma Total:%d", soma)  
    return soma;  
}
```

valor agora é
incrementado
em 2

- ◆ Pode-se colocar qualquer tipo de expressão na parte de incremento do comando *for*

Variações do Comando for

- ◆ Cada expressão no cabeçalho de um laço **for** loop é opcional
- ◆ Se a inicialização é omitida, nenhuma inicialização é feita
- ◆ Se a condição é omitida, a condição é considerada sempre *verdadeira* e o laço continua para sempre (laço infinito)
- ◆ Se o incremento é omitido, nenhuma operação é realizada ao final da iteração do laço

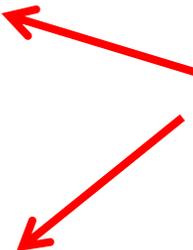
O Comando `for` sem Condição, etc.

```
for (; valor < n; valor++)  
    corpo
```

```
for (; ; valor++)  
    corpo
```

```
for (; ; )  
    corpo
```

Repetição infinita:
cuidado!



Mais Variações do Comando *for*

- ◆ Teste condicional não precisa ser baseado na variável de controle

```
#include <stdio.h>
#include <conio.h>
int main ( ) {
    int i ;
    /* atribui um valor inicial a ch*/
    char ch = 'a' ;
    for(i = 0; ch != 'q' ;i++){
        printf ("\nPasso:%d",i) ;
        ch = getche() ;
    }
}
```

condição não é
baseada na
variável de
controle

Mais Variações do Comando *for*

- ◆ Parte de incremento não precisa incrementar variável de controle
- ◆ Comando *for* pode vir sem corpo

```
#include <stdio.h>
#include <conio.h>
int main ( ) {
    char ch;
    for(ch = getche(); ch != 'q'; ch = getche());
    printf ("\nValor q encontrado");
}
```

Não tem **corpo**

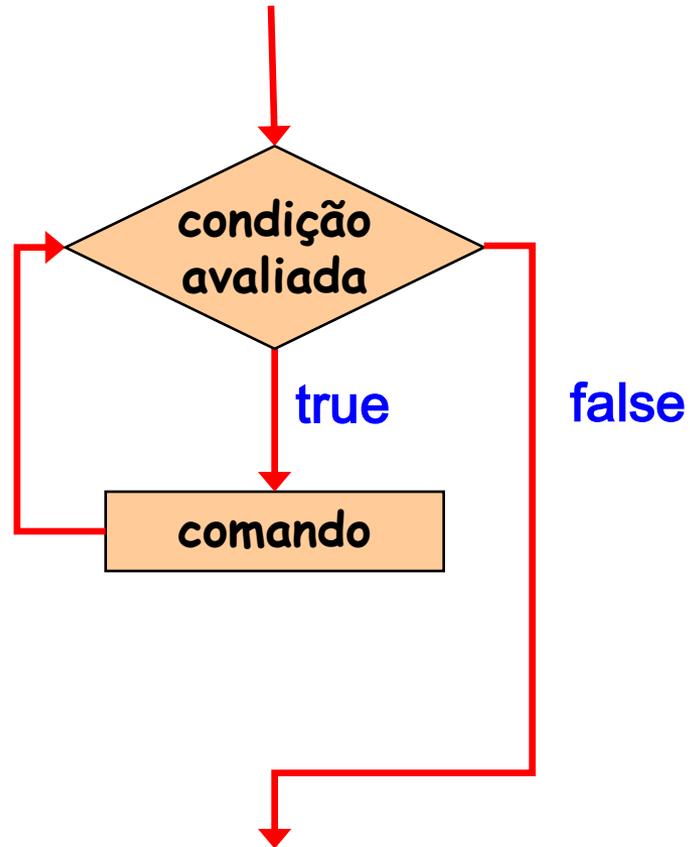
Incremento
pode ser outro
tipo de
expressão

O comando `while`

```
while (condição)  
    corpo
```

- ◆ Executa `corpo` várias vezes até que a avaliação da expressão retorne `false`
- ◆ A condição é avaliada de novo após cada execução de `corpo`
- ◆ Não executa `corpo` nenhuma vez, se de início a avaliação da condição retorna `false`

Fluxo de Controle do Laço while



Entendendo o comando while

```
int somatorio(int n) {  
    int soma = 0;  
    int valor = 0;  
    while ( valor <= n ) {  
        soma = soma + valor;  
        valor++;  
    }  
    return soma;  
}
```

Se **n** for negativo, não se executa o corpo do **while**

É executado quando o **while** termina, quando a condição for **falsa**

Entendendo o comando `while`

```
int somatorio(int n) {  
    int soma = 0;  
    int valor = 0;  
    while ( valor <= n ) {  
        soma = soma + valor;  
        valor++;  
    }  
    return soma;  
}
```

Inicialização da
variável de
controle é feita
fora do laço
while

Incremento da variável
de controle é feita no
corpo do laço **while**

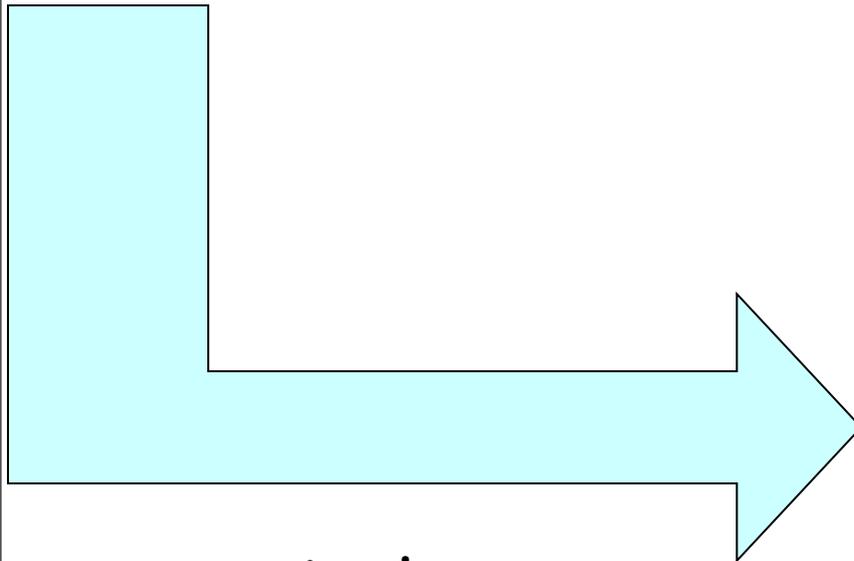
Laço Infinito com o Comando while

```
int somatorio(int n) {  
    int soma = 0;  
    int valor = 0;  
    while ( valor <= n ) {  
        soma = soma + valor;  
    }  
    return soma;  
}
```

Se valor não é incrementado, este comando será executado infinitas vezes

O Comando for e o Comando while

```
for (inicialização ; condição ; incremento)  
    corpo
```

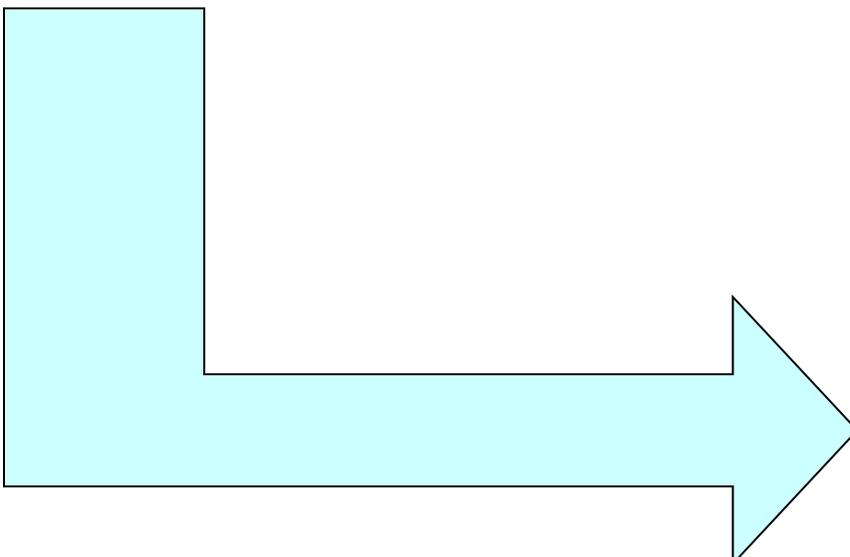


equivale a ...

```
inicialização;  
while (condição) {  
    corpo ;  
    incremento ;  
}
```

O Comando for e o Comando while

```
for (;;)
  corpo
```



```
while (1) {
  corpo;
}
```

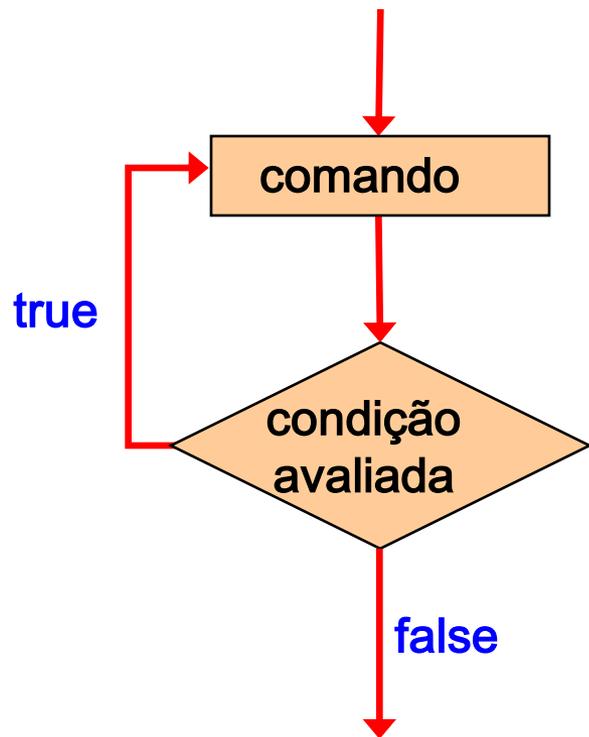
equivale a ...

O Comando do-while

```
do {  
    corpo  
} while (condição)
```

- ◆ Executa **corpo**, pelo menos uma vez, até que a avaliação da condição retorne **false**
- ◆ A condição é avaliada de novo após cada execução de **corpo**

Fluxo de Controle do Laço do-while



Entendendo o comando do-while

```
int somatorio(int n) {  
    int soma = 0;  
    int valor = n;  
    do {  
        soma = soma + valor;  
        valor--;  
    } while ( valor >= 0 )  
  
    return soma;  
}
```

Se **n** for negativo, o corpo do **do-while** é executado pelo menos uma vez

É executado quando o **do-while** termina, quando a condição for **falsa**

Comportamento alterado: **cuidado!**

Os Comandos do-while e while

```
do {  
    corpo  
} while (condição)
```

Equivalente a ...

```
corpo;  
while (condição)  
    corpo;
```

Laços Aninhados

- ◆ Laços podem ser aninhados da mesma forma que comandos condicionais
 - O corpo de um laço pode conter outro laço
- ◆ Para cada iteração do laço externo, o laço interno é completamente executado

Laços Aninhados

```
int somatorioDoSomatorio(int n, int vezes) {  
    int soma, somatorio = 0;  
    int valExt;  
    int valInt;  
    for (valExt = 0; valExt < vezes; valExt++) {  
        valInt = 0; soma = 0;  
        while (valInt <= n) {  
            soma = soma + valInt;  
            valInt++;  
        }  
        somatorio = somatorio + soma;  
    }  
    return somatorio;  
}
```

A cada iteração do **for**,
o laço **while** é
executado

Considerações sobre Laços

- ◆ Os 3 tipos de laços são funcionalmente equivalentes
 - Portanto podem ser usados indiscriminadamente
- ◆ Os laços **for** e **while** são executados 0 ou muitas vezes
- ◆ O laço **do-while** é executado 1 ou muitas vezes

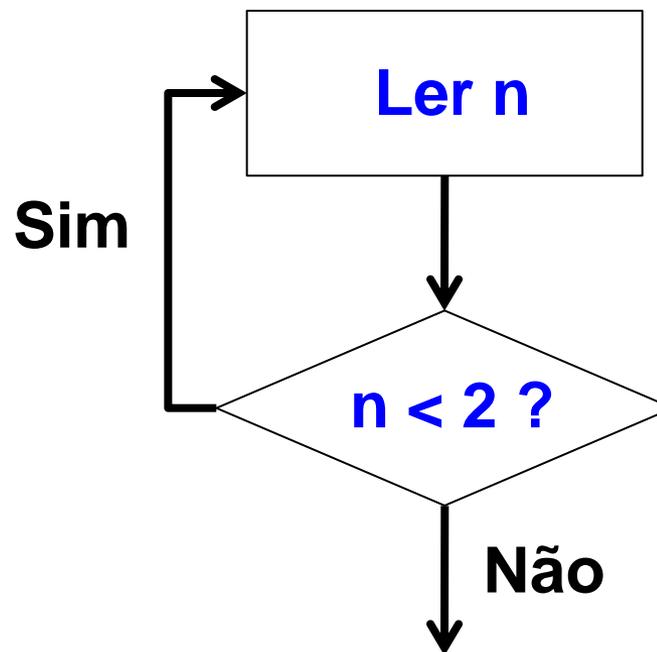
Exemplo de Laços

- ◆ Escreva um programa em C que leia um número inteiro **n** maior do que 1 e informe se o número é primo ou não. Caso o usuário digite um valor menor do que 2, o programa deve pedir novamente para o usuário digitar um valor maior do que 1.

Resolvendo o Problema de Números Primos

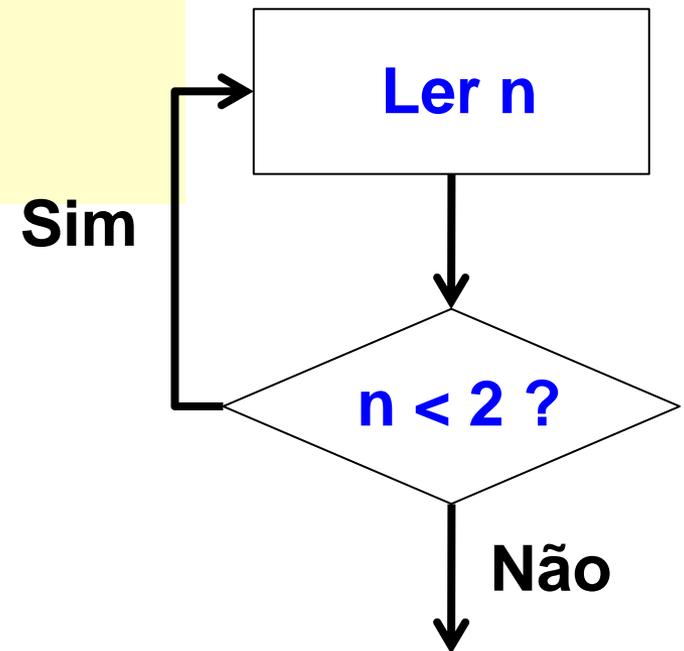
- ◆ 1º Problema: Como forçar o usuário a digitar um valor maior do que 1?

Laço !



Programa para Numeros Primos

```
int main() {  
    int cont = 0, i = 2, n;  
    do {  
        printf("\nDigite numero > 1\n");  
        scanf ("%d",&n);  
    } while ( n < 2 )  
    .....  
}
```

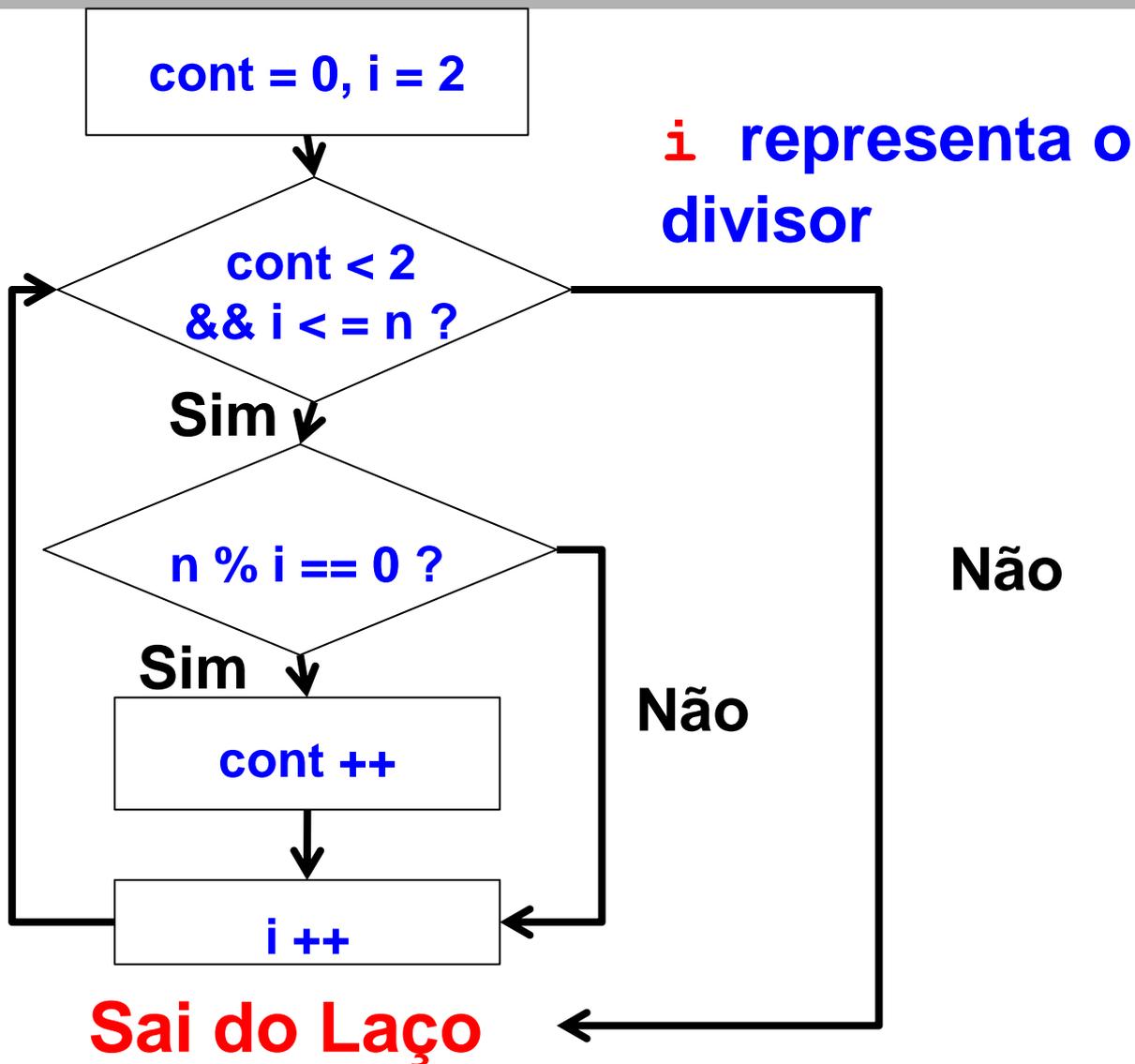


Resolvendo o Problema de Números Primos

- ◆ 2º Problema: Como saber que um número é primo?
 - Ver se n é divisível por mais de um número entre 2 e n

Resolvendo o Problema de Números Primos

Laço !

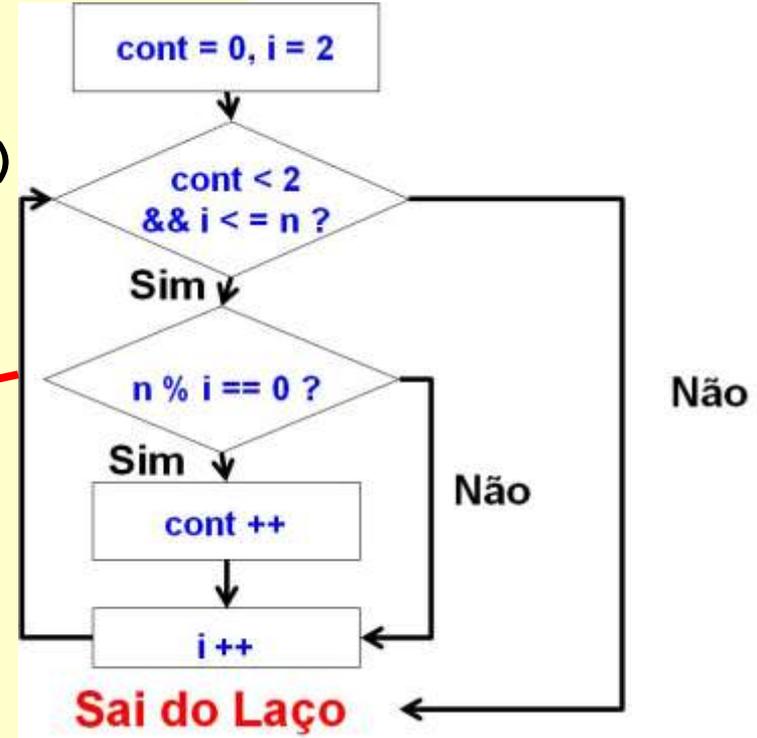


Programa para Numeros Primos

```

int main() {
    int cont = 0, i = 2, n;
    do {
        printf("\nDigite numero > 1\n");
        scanf ("%d",&n);
    } while ( n < 2 )
    while (cont < 2 && i <= n) {
        if ( n % i == 0) {
            cont++;
        }
        i++;
    }
    if (cont > 1) {
        printf("\n %d nao eh primo\n",n);
    } else {
        printf("\n %d eh primo\n",n);
    }
}

```



O Comando `break`

- ◆ Forma Geral do comando `break`

```
break ;
```

- ◆ Tem dois usos distintos

- Para forçar o término de um laço de repetição (*do-while*, *for* ou *while*)
- Para terminar um *case* do comando *switch*

- ◆ Quando o comando `break` é encontrado dentro de um laço de repetição:

- instrução é imediatamente finalizada
- próxima instrução após a estrutura de repetição é executada

- ◆ Deve ser usado com cautela

- Reduz legibilidade
- Pode levar a erros de lógica

O Comando break

```
int somatorio(int n) {  
    int soma = 0;  
    int valor;  
    for (valor = 0; ; valor++) {  
        soma = soma + valor;  
        if (valor == n)  
            break;  
        printf("Soma Parcial:%d\n", soma) ;  
    }  
    printf("Soma Total:%d", soma)  
    return soma;  
}
```

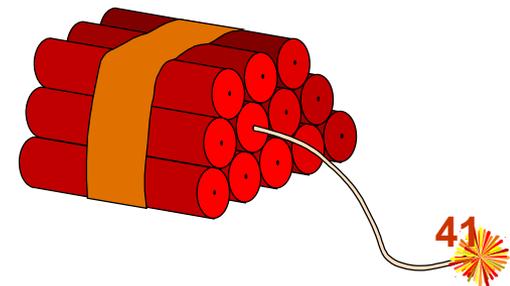
Este comando não
será executado
quando
valor == n

O Comando continue

- ◆ Forma Geral do comando **continue**

continue

- ◆ Termina a execução da iteração atual de um loop (**for, while, do-while**) e volta ao começo deste loop
- ◆ Todos os comandos que seriam executados após o **continue** são descartados



Comando de Desvio - continue

- ◆ Para os comandos *while* e *do-while*, o **continue** causa:
 - a realização imediata do teste da condição correspondente
 - continuidade do processo de repetição dependendo do resultado do teste
- ◆ Para o comando *for*, o **continue** causa:
 - incremento da variável de controle do laço de repetição
 - execução do teste para verificação da condição
 - continuidade do processo de repetição dependendo do resultado do teste

O Comando continue

```
void imprimeNumerosAteCinco() {  
    int valor;  
    for (valor = 0; valor <= 5; valor++) {  
        if (valor == 4)  
            continue;  
        printf("%d ", valor);  
    }  
}
```

Controle pula para o incremento e este comando não será executado quando `valor == 4`

A saída desta função será: 0 1 2 3 5

O Comando continue

```
void imprimeNumerosAteCinco() {  
    int valor = 0;  
    while (valor <= 5) {  
        if (valor == 4)  
            continue;  
        printf("%d", valor);  
        valor++;  
    }  
}
```

Controle pula para o teste e função tem execução infinita quando `valor == 4`

A saída desta função será: 0 1 2 3 ...
laço infinito

Resumindo ...

- ◆ Estruturas de Repetição
 - Necessidade de estruturas de repetição
 - Conceito de laço (*loop*)
 - Tipos de laço em C
 - O comando *for*
 - O comando *while*
 - O comando *do-while*
 - Laços Aninhados
 - Diferenças entre os comandos de repetição
- ◆ Comandos para alterar o fluxo de laços
 - *break e continue*