



# Estruturas de Dados (ED)

**Aula Prática 2 (Parte 1):**

**Comandos de Decisão e Laços (Finito e Infinito)**

**Prof. Daniel Baraldi Sesso**

**[daniel.sesso@docente.unip.br](mailto:daniel.sesso@docente.unip.br)**

## Comandos de decisão

Imagine a seguinte situação: Você está criando um programa para mostrar uma mensagem se a pessoa é idosa ou não

- Se a pessoa tem mais de 60 anos ou mais, mostra a mensagem:  
"Você está na Melhor idade"
- Se a pessoa tem menos de 60 anos, mostra a mensagem:  
"Ainda não viveu nada..."

Como fazer esse programa?



# Comandos de decisão

Uma das principais estruturas em linguagem de programação são as estruturas de decisão, também conhecidas como:

- Condicionais
- Estruturas de controle de fluxo
- Estruturas de seleção
- Estruturas de decisão

Estas estruturas fazem com que algumas instruções em um programa sejam executadas de forma condicional.

## Comandos de decisão: if

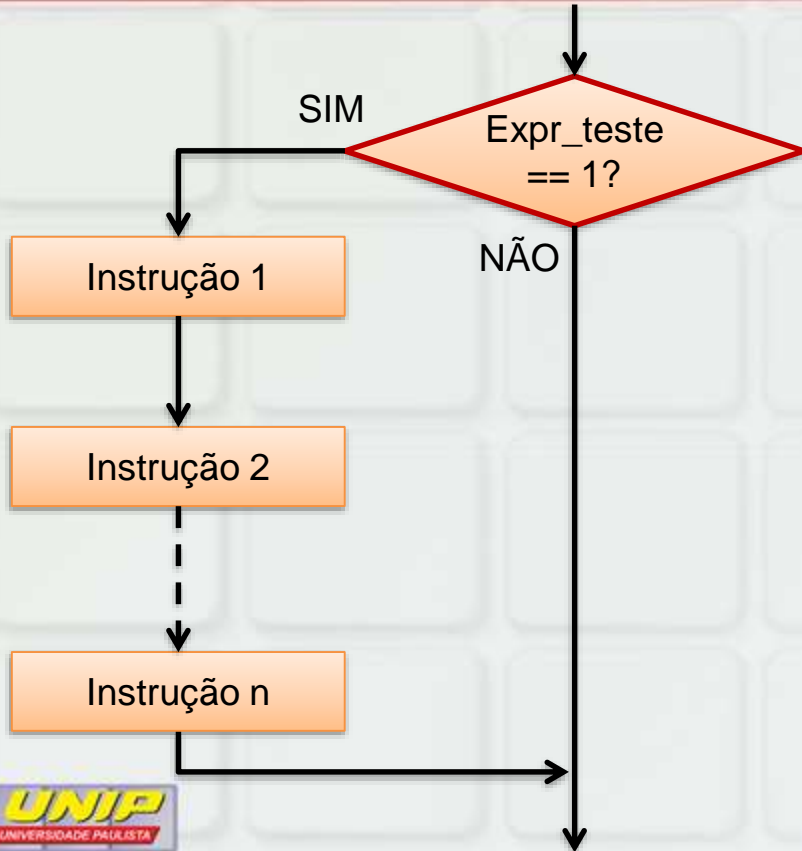
Para poder executar instruções de forma condicional, a linguagem C oferece uma estrutura de decisão simples.

declaração **if** (palavra-chave) seguida de uma expressão de teste  
**se** a expressão de teste for **VERDADEIRA**, **então** executa um conjunto de instruções.

A sintaxe básica do **if** é a seguinte:

```
if (<Expressão de Teste>
{
    <instrução 1>;
    <instrução 2>;
    ...
    <instrução n>;
}
```

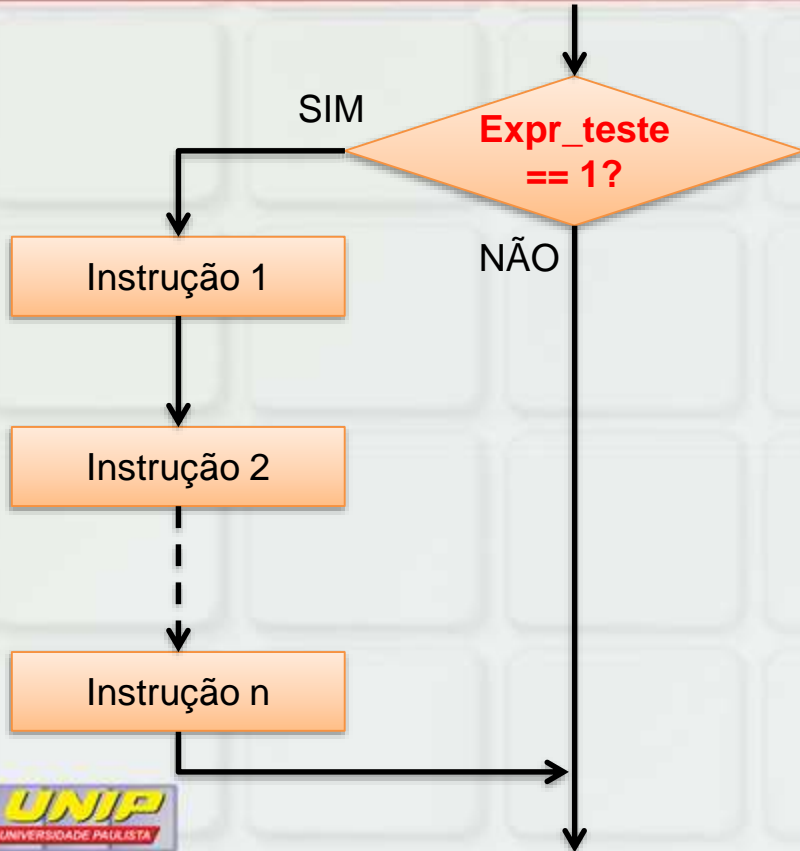
## Comandos de decisão: if



```
if (<Expressão de Teste>)  
{  
    <bloco de código>;  
}
```

**if** é a palavra-chave da linguagem C que inicia a estrutura se decisão  
A única restrição é que deve ser toda escrita em minúsculo

# Comandos de decisão: if

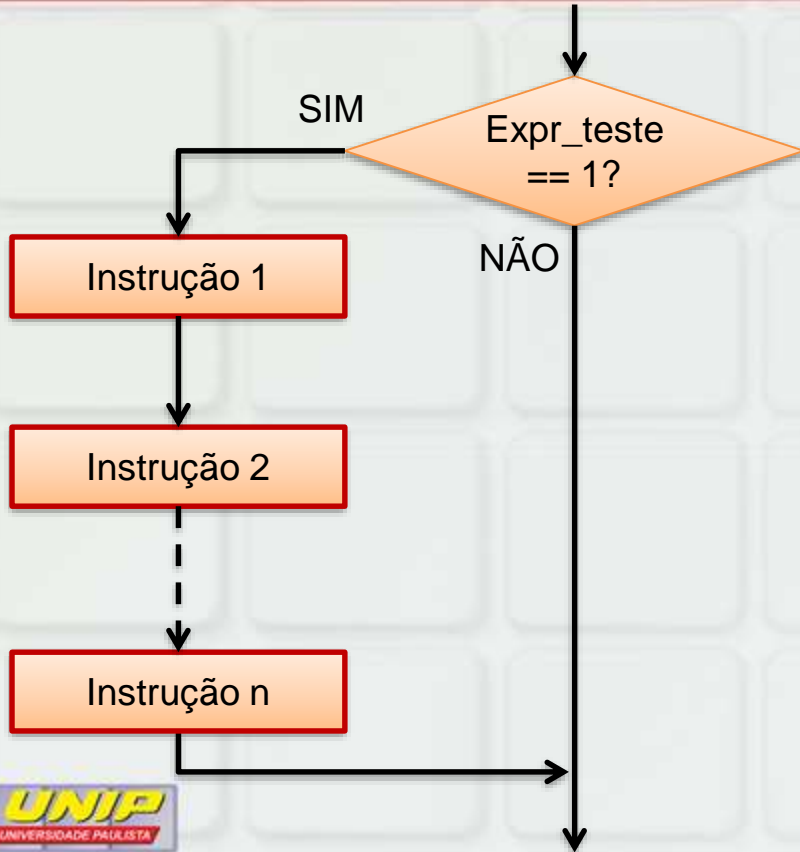


```
if (<Expressão de Teste>)  
{  
    <bloco de código>;  
}
```

Depois do `if` é colocada a expressão de teste que será verificada

O resultado da expressão de teste é um valor 1 (verdadeiro) ou 0 (falso), derivado de uma comparação ou do uso de operadores lógicos

## Comandos de decisão: if



```
if (<Expressão de Teste>
{
    <bloco de código>;
}
```

As instruções do bloco de código que deve ser executado devem estar obrigatoriamente entre chaves.

Se não houver as chaves, somente a primeira instrução será executada caso a expressão de teste seja verdadeira

Após o fim da execução do bloco, o programa volta pra seu fluxo normal

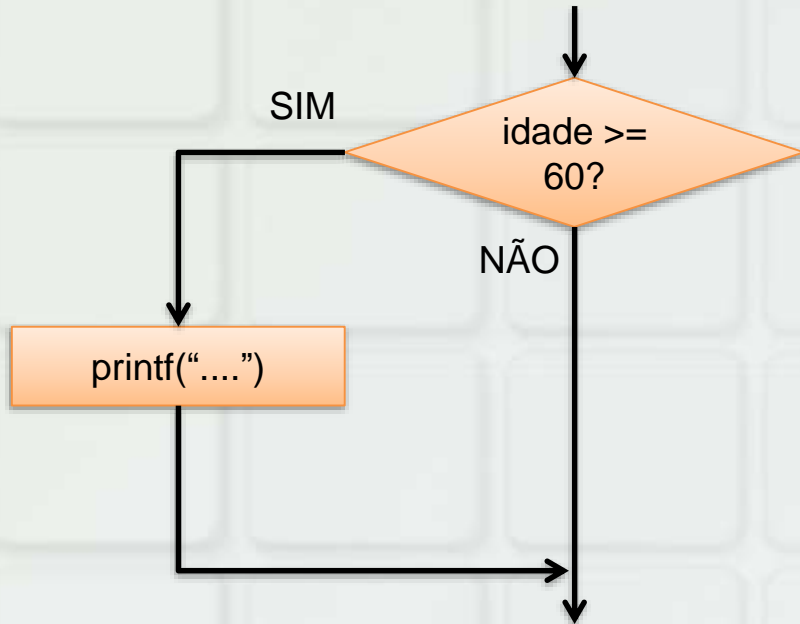
## Comandos de decisão: if

Nesse programa exemplo foi criada uma variável idade com valor 65 e que mostra uma mensagem caso a idade seja maior que 60

```
unsigned int idade = 65;  
  
if (idade >= 60){  
    printf("Voce esta na melhor idade.");  
}
```



## Comandos de decisão: `if`



O que o programa faz?

Testa se idade  $\geq$  60 anos

1 (TRUE)?

Imprime a mensagem

0 (FALSE)?

Continua para a próxima  
instrução depois do bloco de  
código do `if`

## Comandos de decisão: `if`

Nesse programa especificamente, poderíamos ter removido as chaves, pois só há uma instrução a ser executada no bloco de código do comando `if`.

Em havendo mais de uma instrução devemos obrigatoriamente colocar as chaves, pois, caso contrário, somente a primeira instrução será interpretada como estando no bloco do comando `if`.

```
unsigned int idade = 65;  
  
if (idade >= 60)  
    printf("Voce esta na melhor idade.");
```

## Comandos de decisão: `if`

Nesse exemplo os dois comandos são executados, entretanto o segundo comando **não faz parte do bloco `if`**, mas sim do bloco principal do programa

A mensagem será exibida mesmo que o valor da variável idade seja menor que 60 anos.

```
unsigned int idade = 65;

if (idade >= 60)
    printf("Parabens!");
printf("Voce esta na melhor idade.");
```

## Comandos de decisão: if

Esta é outra característica das estruturas de decisão:

Após a execução do bloco condicional, o resto do programa continua executando normalmente.

```
unsigned int idade = 65;  
  
if (idade >= 60)  
    printf("Parabens!");  
printf("Voce esta na melhor idade.");
```

## Comandos de decisão: if

Acabamos de ver como fazer uma estrutura de decisão simples

```
unsigned int idade = 65;  
  
if (idade >= 60){  
    printf("Voce esta na melhor idade.");  
}
```

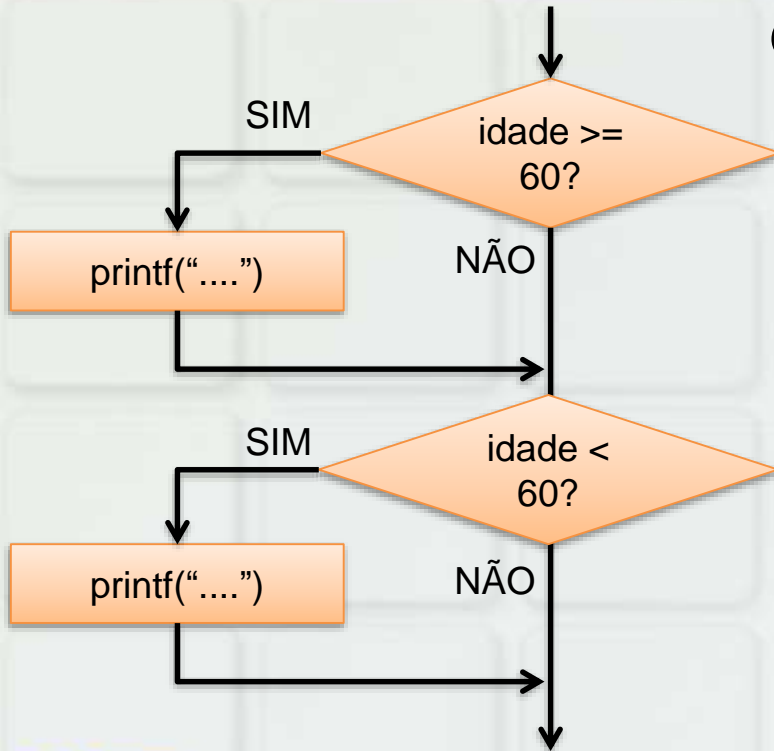
Imagine que você queira testar também se a idade é menos que sessenta, para mostrar a outra mensagem

## Comandos de decisão: if

Basta colocar uma outra estrutura de decisão simples (declaração **if**) com seu bloco de código após a primeira estrutura.

```
unsigned int idade = 65;  
if (idade >= 60){  
    printf("Voce esta na melhor idade!");  
}  
if (idade < 60){  
    printf("Voce ainda nao viveu nada!");  
}
```

# Comandos de decisão: if



O que o programa faz?

Testa se é idade  $\geq 60$

1 (TRUE)?

Imprime a mensagem "... melhor idade..."

0 (FALSE)?

Continua para a próxima instrução

Testa se idade  $< 60$

1 (TRUE)?

Imprime a mensagem "... ainda não viveu..."

0 (FALSE)?

Continua para a próxima instrução

## Comandos de decisão: if

**WARNING!**



A utilização de duas ou mais estruturas de decisão simples para decidir em sequência sobre uma mesma variável pode acarretar em perda de desempenho por tratar em dois momentos sobre a condição da variável

No diagrama anterior, o programa vai verificar as duas condições, independente do resultado da primeira

Entretanto, a segunda condição só vai ser verdadeira se a primeira for falsa

1: idade  $\geq$  60

2: idade  $<$  60

**Não existe outra possibilidade!**



## Comandos de decisão: **if-else**

Para estes casos, em que as condições verificadas são excludentes (ou a 1ª é verdadeira ou a 2ª é verdadeira), existe uma estrutura de decisão composta, utilizando uma declaração **else**

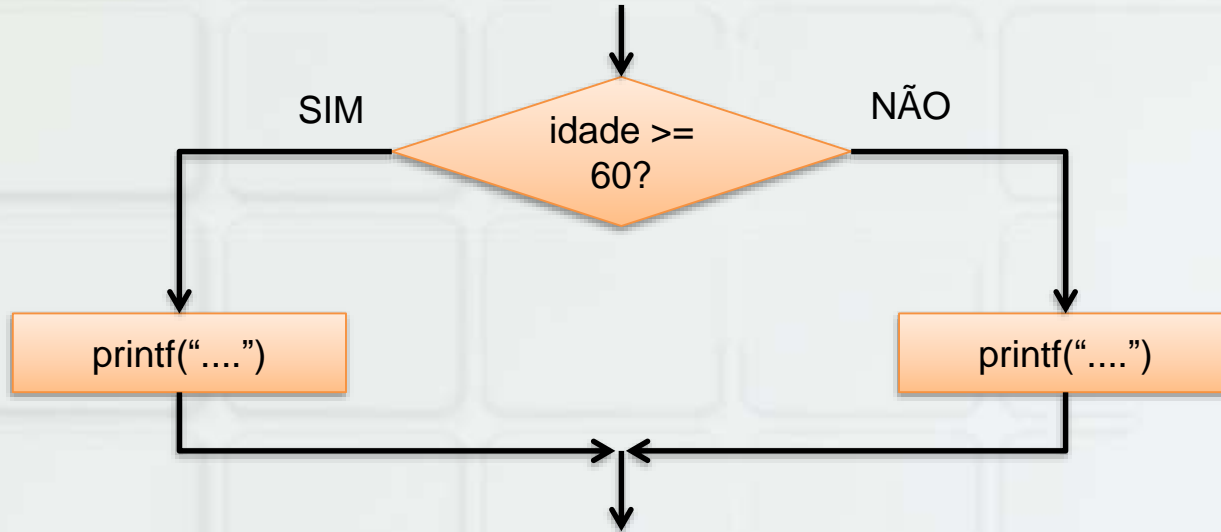
- Executa uma coisa ou outra
- Não existe outra possibilidade para o valor da variável
- Evita uma operação de comparação em seu programa

O **else** é uma “continuação” do **if**

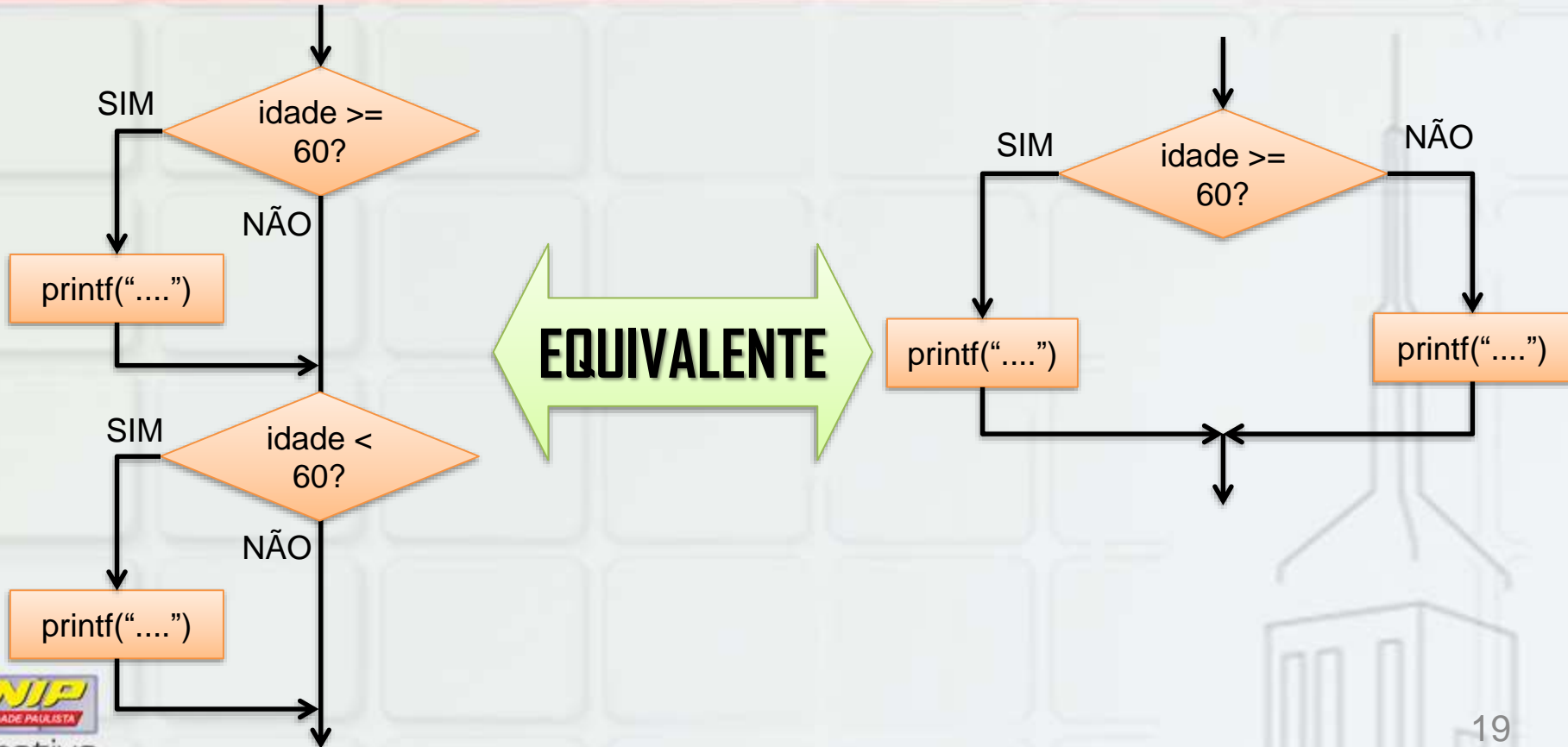
- Só pode ser utilizado se existir um **if**
- Conhecido como uma declaração **if-else**
- **Se** uma condição for verdadeira, **então** execute um conjunto de instruções;  
**senão**, execute outro conjunto

## Comandos de decisão: if-else

Veja que agora há **apenas uma verificação**, e ambos os lados continuam a execução do programa no mesmo ponto



# Comandos de decisão: if e if-else



## Comandos de decisão: `if-else`

A sintaxe de uma declaração `if-else` é a seguinte:

Uma única instrução (sem chaves)

```
if (<Expressão de Teste>
    <instrução do bloco if>;
else
    <instrução do bloco else>;
```

- Se a condição for verdadeira, executa bloco de código `if`
- Senão, executa o bloco de código `else`

Várias instruções (com chaves)

```
if (<Expressão de Teste>)
{
    <instrução do bloco if>;
    <instrução do bloco if>;
}
else
{
    <instrução do bloco else>;
    <instrução do bloco else>;
}
```

## Comandos de decisão: `if-else`

Lembre-se que:

- A declaração `else` só pode existir se existir uma declaração `if`
- A declaração do `else` é opcional; pode existir `if` sem um `else` (decisão simples)
- A declaração `else` não tem um teste lógico; ela é executada sempre que o teste do `if` for '0' (FALSO)

## Comandos de decisão: `if-else`

No exemplo de verificar a idade, a declaração `if-else` ficará assim:

```
unsigned int idade = 65;

if (idade >= 60)
{
    printf("Voce esta na melhor idade!");
}
else
{
    printf("Voce ainda nao viveu nada!");
}
```

## Comandos de decisão aninhados

E se fosse necessário fazer mais uma comparação?

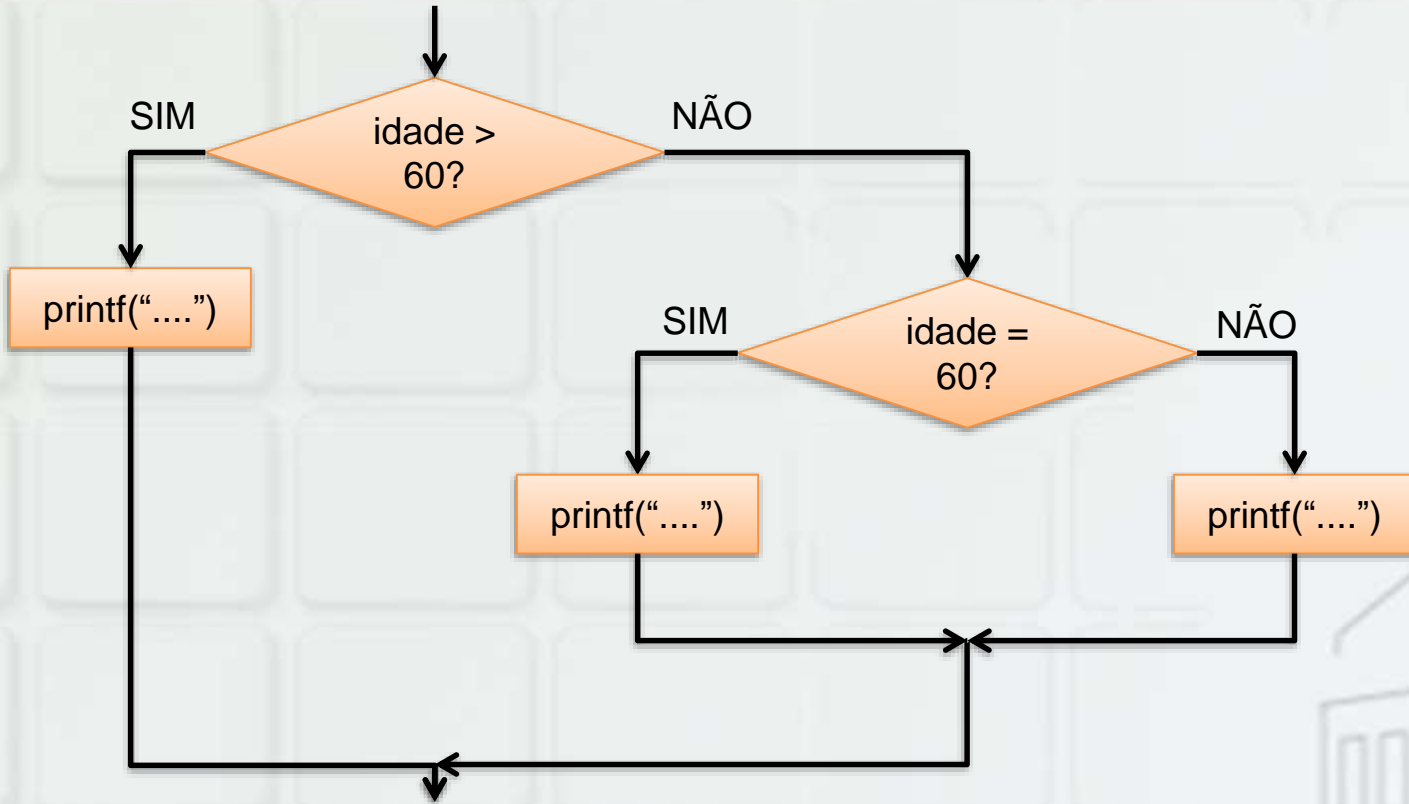
Por exemplo:

Se idade maior que 60, então mostre uma mensagem X  
Senão, se idade igual a 60, então mostre uma mensagem Y  
Senão, mostre uma mensagem Z

Existem três ou mais condições de um mesmo valor para testar



# Comandos de decisão aninhados





## Comandos de decisão aninhados

Para avaliar três ou mais condições de um valor, em C utiliza-se a construção conhecida como **else-if**

**Se** uma condição for verdadeira, **então** execute um conjunto de instruções;  
**Senão, se** uma segunda condição for verdadeira, então execute outro conjunto de instruções  
**Senão**, execute outro conjunto de instruções

**else-if** é uma continuação do **if**  
Só pode ser utilizado se existir um **if**

# Comandos de decisão aninhados

## Sintaxe da construção **else-if**

- Se expressão 1 for verdadeira ('1'), executa bloco de código **if**
- Senão, se expresssão 2 for verdadeira, executa bloco de código **else-if**
- Senão, executa bloco de código **else**

```
if (<Expressão de Teste 1>
{
    <instrução do bloco if>;
    <instrução do bloco if>;
}
else
    if (<Expressão de Teste 2>)
    {
        <instrução do bloco else-if>;
        <instrução do bloco else-if>;
    }
    else
    {
        <instrução do bloco else>;
        <instrução do bloco else>;
    }
```

# Comandos de decisão aninhados

Lembre-se que:

- A declaração **else if** só pode existir se existir uma declaração **if**
- A declaração do **else if** é opcional; pode existir **if** sem um **else if** (decisão simples)
- **else** (no final) também continua sendo opcional
- A declaração **else if** tem um teste lógico assim como **if**; ela é executada sempre que o teste do **if** for '0' (FALSO)

## Comandos de decisão aninhados

No exemplo de verificar a idade, usando a construção **else-if** ficará assim:

```
unsigned int idade = 65;
if (idade > 60){
    printf("Voce esta na melhor idade!");
}
else if (idade == 60){
    printf("Voce esta no limite!");
}
else{
    printf("Voce ainda nao viveu nada!");
}
```

## Comandos de decisão aninhados

Na construção **else-if** podem existir quantas declarações **else-if** forem necessárias

Veja o exemplo a seguir:

Faça um programa onde o usuário coloque sua idade e mostre mensagens de acordo com o idade:

- Se a idade for maior ou igual a 60, escreve: “Idoso”
- Se a idade for entre 18 (inclusive) e 60, escreve “Adulto”
- Se a idade for entre 13(inclusive) e 18, escreve “Adolescente”
- Se a idade for entre 3 (inclusive) e 13, escreve “Criança”
- Qualquer outro caso, escreve “Bebê”

# Comandos de decisão aninhados

```
#include <stdio.h>
int main(void) {
    unsigned int idade = 0;
    printf("Qual sua idade? Idade= ");
    scanf("%d", &idade);
    if (idade >= 60)
        printf("Idoso");
    else if (idade >= 18 && idade < 60)
        printf("Adulto");
    else if (idade >= 13 && idade < 18)
        printf("Adolescente");
    else if (idade >= 3 && idade < 13)
        printf("Criança");
    else
        printf("Bebe");
    return 0;
}
```

```
#include <stdio.h>
int main(void) {
    unsigned int idade = 0;
    printf("Qual sua idade? Idade= ");
    scanf("%d", &idade);
    if (idade >= 60){
        printf("Idoso");
    }
    else{
        if (idade >= 18 && idade < 60){
            printf("Adulto");
        }
        else{
            if (idade >= 13 && idade < 18){
                printf("Adolescente");
            }
            else{
                if (idade >= 3 && idade < 13){
                    printf("Criança");
                }
                else{
                    printf("Bebe");
                }
            }
        }
    }
    return 0;
}
```

# Comando switch

- O comando switch permite selecionar uma entre várias ações alternativas.
- Embora construções **if-else** possam executar testes para escolha de uma entre várias alternativas, muitas vezes se tornam “*deselegantes*” (e em muitos casos, confuso de entender a lógica do que está programado)
  - ✓ Dificulta também a depuração do código.
- O comando **switch** tem um formato mais limpo e claro se comparado com o **if-else** (comando de decisão aninhado).

# Comando switch

Sem ponto-e-vírgula

- O comando **switch** consiste na palavra-chave **switch** seguida do nome de uma variável ou de um valor numérico constante entre parênteses.





```
switch (<variável ou constante>)  
{  
    case <constante1>:  
        <instrução>;  
        <instrução>;  
        break;  
    case <constante2>:  
        <instrução>;  
        <instrução>;  
        break;  
    case <constante3>:  
        <instrução>;  
        <instrução>;  
        break;  
    default:  
        <instrução>;  
        <instrução>;  
}
```



# Comando switch

Dois pontos

- O corpo do comando é composto de vários **casos** que devem ser rotulados com uma **constante** e, opcionalmente, um caso **default**.

```
switch (<variável ou constante>)  
{  
    case <constante1>:   
        <instrução>;  
        <instrução>;  
        break;  
    case <constante2>:   
        <instrução>;  
        <instrução>;  
        break;  
    case <constante3>:   
        <instrução>;  
        <instrução>;  
        break;  
    default:   
        <instrução>;  
        <instrução>;  
}  

```

# Comando switch

- A expressão entre parênteses após a palavra-chave switch determina para qual caso será desviado o controle do programa.

```
switch (<variável ou constante>
{
    case <constante1>:
        <instrução>;
        <instrução>;
        break;
    case <constante2>:
        <instrução>;
        <instrução>;
        break;
    case <constante3>:
        <instrução>;
        <instrução>;
        break;
    default:
        <instrução>;
        <instrução>;
}
```

# Comando switch

- O corpo de cada caso é composto por qualquer número de instruções.

```
switch (<variável ou constante>)  
{  
    case <constante1>:  
        <instrução>;  
        <instrução>;  
        break;  
    case <constante2>:  
        <instrução>;  
        <instrução>;  
        break;  
    case <constante3>:  
        <instrução>;  
        <instrução>;  
        break;  
    default:  
        <instrução>;  
        <instrução>;  
}
```

# Comando switch

- Geralmente, a última instrução é **break**, o que causa a saída imediata de todo o corpo do **switch**.
- Na falta do comando **break**, todas as instruções, a partir do caso escolhido até o término do comando, serão executadas, mesmo sendo pertencentes aos casos seguintes.
- O comando **break** tem somente dois usos em C: em laços ou no comando **switch**.

```
switch (<variável ou constante>)  
{  
    case <constante1>:  
        <instrução>;  
        <instrução>;  
        break;  
    case <constante2>:  
        <instrução>;  
        <instrução>;  
        break;  
    case <constante3>:  
        <instrução>;  
        <instrução>;  
        break;  
    default:  
        <instrução>;  
        <instrução>;  
}
```

# Comando switch: Exemplo (Calculadora)

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

int main()
{
    const int TRUE=1;

    while(TRUE)/* Sempre verdadeiro*/
    {
        float n1,n2;
        char op;

        printf("\nDigite número operador número: ");
        scanf("%f%c%f", &n1, &op, &n2);

        if( n1 == 0.0 ) break;/*Termina se Zero digitado*/
```

CONTINUA...

# Comando switch: Exemplo (Calculadora)

## CONTINUAÇÃO...

```
switch(op)
{
case '+':
    printf("\n%f", n1 + n2);
    break;

case '-':
    printf("\n%f", n1 - n2);
    break;

case '*':
    printf("\n%f", n1 * n2);
    break;

case '/':
    printf("\n%f", n1 / n2);
    break;

default:
    printf("\nOperador desconhecido.");
}

printf("\n");
system("PAUSE");
return 0;
}
```

## Estruturas de repetição (laços)

- **Estruturas de repetição**, também conhecidas como **laços** (em inglês **loop**) ou **controle iterativo**, são estruturas que possibilitam a execução de um comando diversas vezes sem a necessidade de escrever um comando várias vezes
- Um comando é **executado até que um critério de parada** (**expressão de teste**) seja satisfeito
  - ❑ O critério de parada é o resultado da expressão de teste ser '1' (TRUE) ou '0' (FALSE)

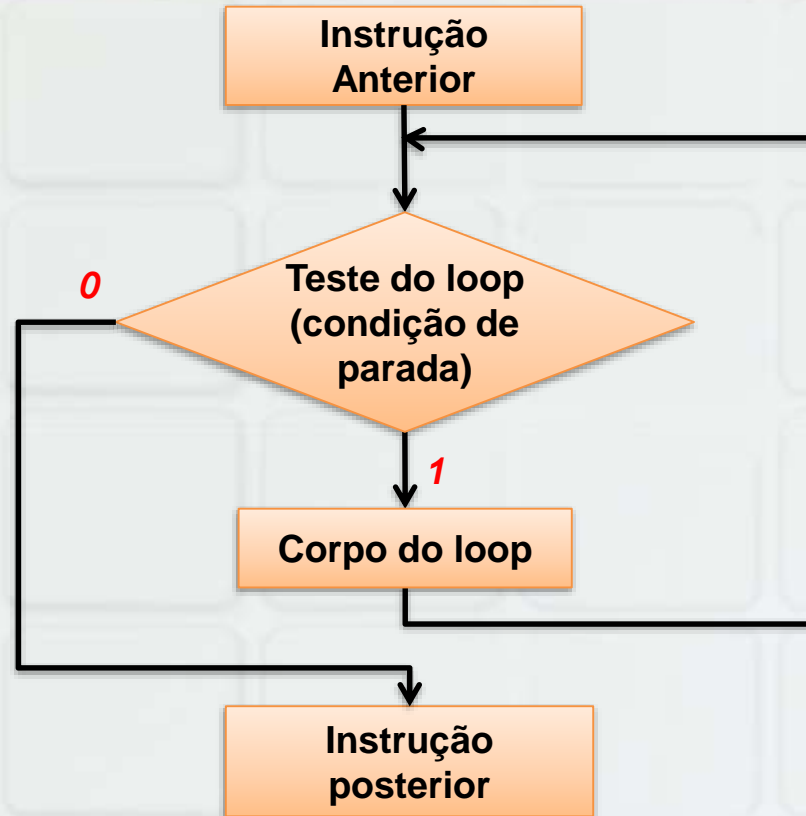
## Estruturas de repetição (laços)

A linguagem C classifica as estruturas de controle iterativo em:

- **loop definido** ou **loop for** quando se sabe previamente o número de vezes que um bloco de instruções será executado.
- **Loop indefinido** ou **loop while** quando o bloco de instruções será executado até que uma condição se torne verdadeira ou falsa.



# Estruturas de repetição (laços)



## Estruturas de repetição (laços)

Por exemplo, para mostrar o números de 1 a 100, utilizando comandos sequenciais, seria necessário escrever 100 comandos **printf()**, com os números de 1 a 100

```
printf("%d",1);  
printf("%d",2);  
printf("%d",3);  
printf("%d",4);  
printf("%d",5);  
...  
printf("%d",100);
```

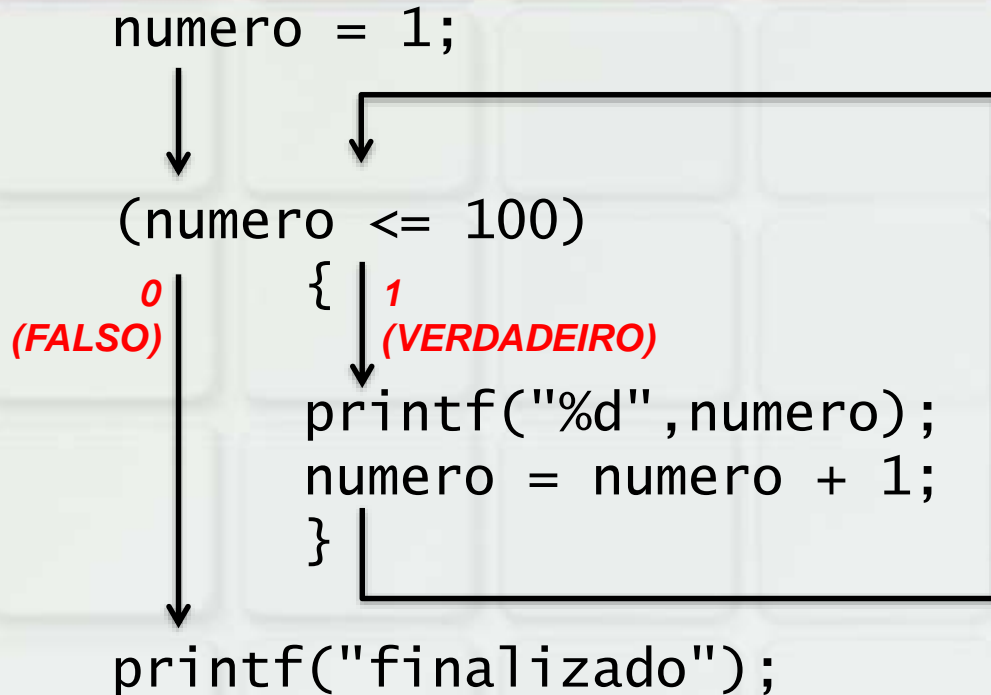
**Existe uma forma mais simples e rápida de fazer isso!**

## Estruturas de repetição (laços)

Na forma geral a regra seria a seguinte:

- Inicie uma variável começando em 1 (variável de controle)
- Teste se a variável é menor ou igual a 100
- Enquanto a condição acima for verdadeira:
  - Mostre o número da variável
  - Some 1 a variável inicial
- Quando a condição não for mais verdadeira (variável maior que 100), mostre um mensagem de finalizado

## Estruturas de repetição (laços)



Repare que a cada iteração do *loop*, o valor de **numero** é alterado

# O laço for

O laço **for** é geralmente usado quando queremos repetir algo por um número fixo de vezes.

Isso significa que utilizamos um laço **for** quando **sabemos previamente o número de vezes a repetir.**

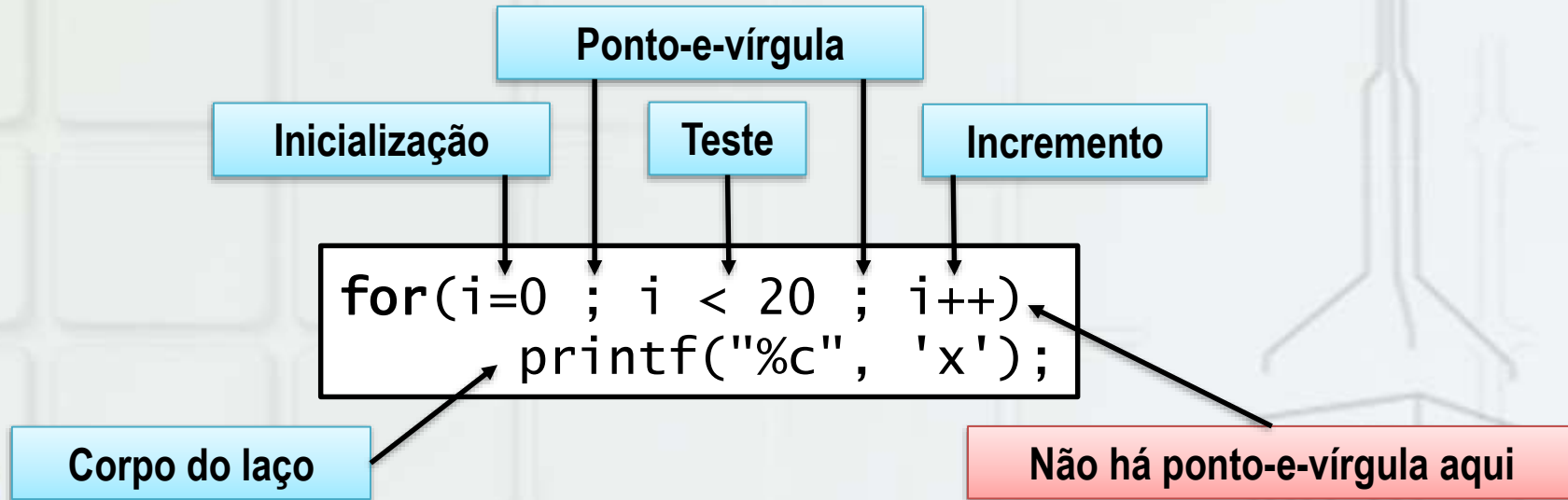
```
#include <stdio.h>
int main()
{
    int i;
    for( i=0 ; i < 20 ; i++ ) /* Imprime 20 * */
        printf("%c" , '*');

    printf("\n");
    return 0;
}
```

# Sintaxe do laço for

A sintaxe consiste na palavra-chave `for` seguida de parênteses que contêm três expressões separadas por ponto-e-vírgulas: a primeira é chamada de **inicialização**, a segunda de **teste** e a terceira de **incremento**.

Qualquer uma delas pode conter qualquer instrução válida em C.



## Sintaxe do laço for: Operador vírgula

Qualquer uma das expressões de um laço **for** pode conter várias instruções separadas por vírgulas.

- A vírgula, nesse caso, é um operador C que significa “faça isso e depois isso”.
- Um par de expressões separadas por vírgula é avaliado da esquerda para a direita.

```
int i, j;  
for(i = 0, j = 1 ; (i + j) < 100 ; i++, j++)  
    printf("%d", i + j);
```

# Sintaxe do laço for: Exemplos de uso (1)

## Usando caracteres:

- A variável do laço pode ser do tipo **char**:

```
char ch;  
for(ch = 'a'; ch <= 'z'; ch++)  
    printf("\nO valor ASCII de %c é %d", ch, ch);
```

## Usando chamadas a funções:

É possível chamar funções de dentro de expressões do laço **for**.

```
unsigned char ch;  
for(ch = getch(); ch != 'X'; ch = getch())  
    printf("%c", ch + 1);
```



## Sintaxe do laço for: Omitindo expressões

Qualquer uma das três expressões de um laço for pode ser omitida, embora os ponto-e-vírgulas devam permanecer.

Se a expressão de inicialização ou a de incremento for omitida, **será simplesmente desconsiderada**.

Se a condição de teste não estiver presente, **será considerada permanentemente verdadeira**.

```
unsigned char ch;  
for( ; (ch = getch()) != 'X'; )  
    printf("%c", ch + 1);
```

**Nota:** coloca-se parênteses extras envolvendo a expressão de atribuição (**ch = getch()**). Esses parênteses são realmente necessários, pois a precedência de **!=** é maior que a de **=**; isso significa que, sem os parênteses, o teste relacional **!=** será feito antes da atribuição.

## Sintaxe do laço for: Exemplos de uso (2)

### Laço infinito:

Um laço infinito é aquele que é executado sempre, sem parar. Ele sempre tem a expressão de teste verdadeira, e **um modo de parar sua execução é desligando o computador**.

```
for( ; ; )  
    printf("Laço infinito");
```

### Omitindo o corpo do laço for:

O corpo do laço pode ser vazio; entretanto, o ponto-e-vírgula deve permanecer.

```
for( ; (ch = getch()) != 'X'; printf("%c", ch + 1));
```

```
for( i = 0; i <= 1000; i = i + 1);
```

## Sintaxe do laço for: Múltiplas instruções no corpo do laço

Se um laço for deve executar varias instruções a cada iteração, elas **precisam estar entre chaves.**

```
for(i = 0; i < 10 ; i++)  
{  
    <instrução 1>;  
    <instrução 2>;  
    ...  
    <instrução n>;  
}
```

# Laços for aninhados

Quando um laço for faz parte do corpo de outro laço for, dizemos que o laço interno está aninhado.

```
#include <stdio.h>
int main(){
    int i,j,k;
    system("cls");/*Limpa a tela */

    for(k=0; k<=1 ; k++){
        printf("\n");
        for(i=1 ; i <= 4 ; i++)
            printf("TABUADA DO %3d      ", i+4*k+1);
        printf("\n");

        for( i = 1; i <= 9 ; i++){
            for( j=2+4*k; j <= 5+4*k; j++)
                printf("%3d x%3d = %3d      ", j,i,j*i);
            printf("\n");
        }
    }
    return 0;
}
```

# O laço `while`

O segundo comando de laço em C é o **`while`** (que significa “*enquanto*”). O laço **`while`** parece simples se comparado ao laço **`for`**; ele utiliza os mesmos elementos, mas estes são distribuídos de maneira diferente no programa.

Utilizamos o laço **`while`** quando o laço **pode ser terminado de forma inesperada**, por condições desenvolvidas dentro do corpo do laço.

```
#include <stdio.h>
#include <conio.h> /* para getch() */
int main()
{
    int cont=0; /* Contador */

    while(getch() != '\r') /* Enquanto não [Enter] */
        cont++; /* Corpo do laço */
    /* Fora do laço */
    printf("\nO número de caracteres é %d\n" , cont);
    return 0;
}
```

# Sintaxe do laço **while**

O comando **while** consiste na palavra-chave **while** seguida de uma **expressão de teste** entre parênteses.

Se a expressão de teste for verdadeira, o corpo do laço é executado uma vez e a expressão de teste é avaliada novamente.

```
<inicialização>;  
while(<Teste>){  
    ...  
    <instrução>;  
    <incremento>;  
    ...  
}
```

## Sintaxe do laço while

Esse ciclo de teste e execução é repetido **até que** a expressão de teste se torne **falsa (igual a zero)**, então o laço termina e o controle do programa passa para a linha seguinte ao laço.

O corpo de um **while** pode ter uma única instrução terminada por ponto-e-vírgula, varias instruções entre chaves ou ainda nenhuma instrução, mantendo o ponto-e-vírgula.

```
<inicialização>;  
while(<Teste>){  
    ...  
    <instrução>;  
    <incremento>;  
    ...  
}
```

## Sintaxe do laço `while`

Em geral, um laço `while` pode substituir um laço `for` da seguinte forma dentro de um programa:

```
<inicialização>;  
while(<Teste>){  
    ...  
    <instrução>;  
    <incremento>;  
    ...  
}
```



# O laço do-while

O terceiro e último comando de laço em C é o laço **do-while**.

Esse laço é bastante similar ao laço **while**.

Ele é utilizado em situações em que é necessário **executar o corpo do laço uma primeira vez e, depois, avaliar a expressão de teste** e criar um ciclo repetido.

Quando usar a estrutura **do-while**?



## Usando o laço do-while

Várias razões influem na consideração de laços que avaliam a expressão de teste antes de serem executados como superiores:

1. Legibilidade, isto é, ler a expressão de teste antes de percorrer o laço ajuda o leitor a interpretar facilmente o sentido do bloco de instruções.
2. Possibilidade da execução do laço mesmo que o teste seja falso de início.

## Sintaxe do laço `do-while`

O comando **`do-while`** consiste na palavra-chave `do` seguida de um bloco de uma ou mais instruções entre chaves e terminada pela palavra-chave **`while`** seguida de uma expressão de teste entre parênteses terminada por ponto-e-vírgula.

```
do
{
    <instrução>;
    <instrução>;
    ...
} while(<Teste>);
```

Ponto-e-vírgula aqui

## Sintaxe do laço do-while

Primeiramente, o bloco de código é executado (1); em seguida, a expressão de teste entre parênteses é avaliada (2); se verdadeira, o corpo do laço é mais uma vez executado (3) e a expressão de teste é avaliada novamente (4).

Esse ciclo de execução do bloco e teste é repetido até que a expressão de teste se torne falsa (igual a zero), quando o laço termina e o controle do programa passa para a linha seguinte ao laço.

```
do
{
    <instrução>; 1
    <instrução>;
    ...
} while(<Teste>; 2
```

## Sintaxe do laço do-while

Primeiramente, o bloco de código é executado (1); em seguida, a expressão de teste entre parênteses é avaliada (2); se verdadeira, o corpo do laço é mais uma vez executado (3) e a expressão de teste é avaliada novamente (4).

Esse ciclo de execução do bloco e teste é repetido até que a expressão de teste se torne falsa (igual a zero), quando o laço termina e o controle do programa passa para a linha seguinte ao laço.

```
do
{
    <instrução>; 3
    <instrução>;
    ...
} while(<Teste>; 4
```

```
#include <stdio.h>
#include <stdlib.h> /* para system() e rand() */
#include <conio.h> /* para getch() */
int main()
{
    char resp; /* resposta do usuário */
    char secreto;
    int tentativas;

    do /*inicio do laço */
    {
        secreto = rand() % 26 + 'a';
        tentativas = 1;
        printf("\n\nDigite uma letra entre 'a' e 'z':\n");

        while((resp=getch())!= secreto)
        {
            printf("%c é incorreto. Tente novamente\n",resp);
            tentativas++;
        }
        printf("%c É CORRETO!!\n", resp);
        printf("Voce acertou em %d tentativas\n", tentativas);
        printf("\nQuer jogar novamente? (s/n): ");
    }while(getche() == 's');
    printf("\nAte logo e boa sorte!\n");
    system("PAUSE");
    return 0;
}
```

## Mais sobre blocos de código

```
{  
Bloco 1;  
  {  
    Bloco 2;  
    Bloco 2;  
      {  
        Bloco 3;  
      }  
    Bloco 2;  
  }  
Bloco 1;  
Bloco 1;  
}
```

Blocos de código são pedaços de código delimitados por chaves e que realizam um conjunto de tarefas específico.

Pode haver vários blocos de código dentro de um programa, e também blocos dentro de blocos (conhecido como blocos aninhados)

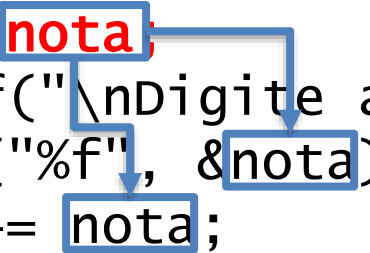
Veja o exemplo ao lado para entender os blocos aninhados

## Visibilidade de variáveis dentro e fora do bloco

Um aspecto importante dos blocos de código é o de que uma variável declarada dentro de um bloco não é visível fora dele.

No programa, declaramos a variável `nota` dentro do bloco de código do laço `for`. Essa variável só pode ser acessada pelas instruções desse mesmo bloco e pelas que estão escritas **após sua declaração**.

```
for( i=0; i < max ; i++){  
    float nota;  
    printf("\nDigite a nota %d : " , i+1);  
    scanf("%f", &nota);  
    soma += nota;  
}
```





## Criando blocos dentro de blocos

Em todo lugar onde é possível colocar uma instrução C, é também possível inserir um bloco de código.

```
#include <stdio.h>
int main()
{
    int i=5;
    /* Início do bloco */
        int i=150;
        printf("%d\n", i); /* Imprime 150 */

    /* Fim do bloco */

    printf("%d\n", i); /* Imprime 5 */
    return 0;
}
```

## Criando blocos dentro de blocos

Observe a variável `i` do bloco interno no programa.

Essa é uma nova variável com o mesmo nome da variável criada no bloco da função `main()`.

```
#include <stdio.h>
int main()
{
    int i=5;
    /* Início do bloco */
        int i=150;
        printf("%d\n", i); /* Imprime 150 */

    /* Fim do bloco */

    printf("%d\n", i); /* Imprime 5 */
    return 0;
}
```

## Criando blocos dentro de blocos

Ela é criada quando o bloco inicia sua execução e é destruída quando o bloco termina. Assim, a instrução após o fim do bloco utiliza a variável `i` do bloco de `main()`, pois a outra não mais existe.

```
#include <stdio.h>
int main()
{
    int i=5;
    /* Início do bloco */
    int i=150;
    printf("%d\n", i); /* Imprime 150 */

    /* Fim do bloco */

    printf("%d\n", i); /* Imprime 5 */
    return 0;
}
```

# Dúvidas



**BONS ESTUDOS!**

