



# **LÓGICA DE PROGRAMAÇÃO PARA ENGENHARIA**

## **OUTRAS ESTRUTURAS DE REPETIÇÃO**

Prof. Dr. Daniel Caetano

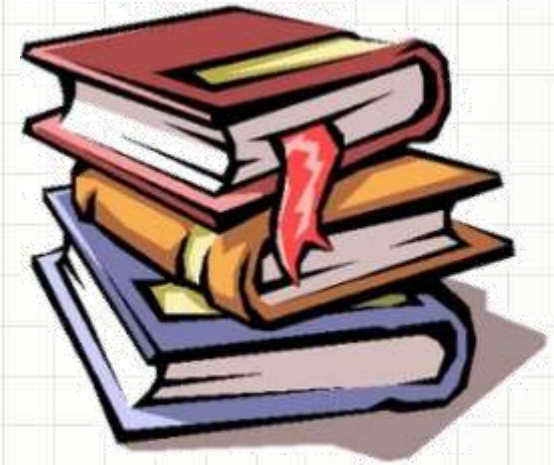
2012 - 2

# Objetivos

- Conhecer as várias estruturas de repetição da linguagem C/C++
- Compreender o uso de cada uma destas estruturas
- Capacitar para a criação de algoritmos que envolvam repetição
- **PARA CASA**
  - Lista de Exercícios 2 está **ONLINE!**



# Material de Estudo



---

## Material

## Acesso ao Material

Notas de Aula

<http://www.caetano.eng.br/>  
(Aula 14)

Apresentação

<http://www.caetano.eng.br/>  
(Aula 14) – PARCIAL / COMPLETO

Material Didático

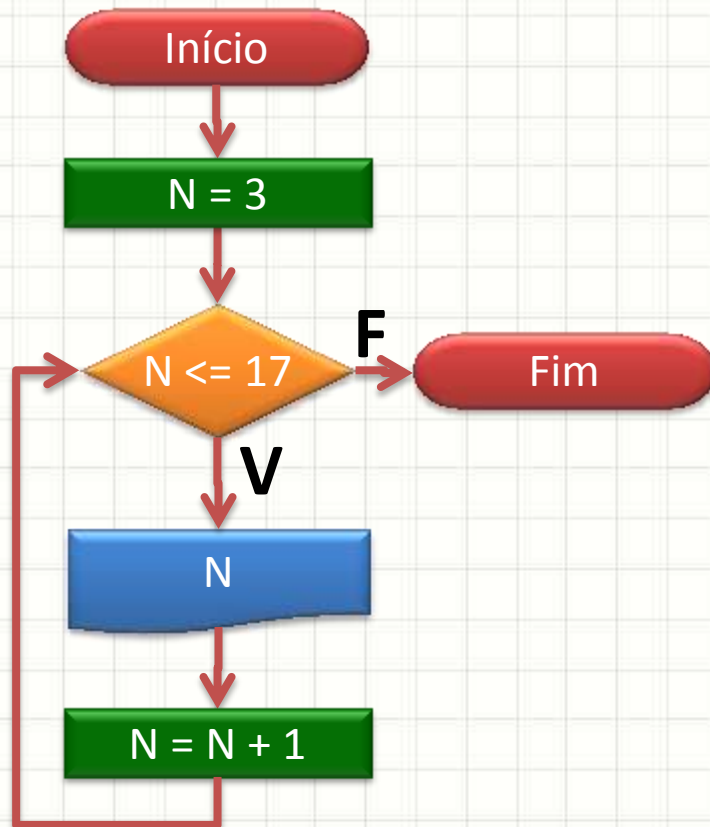
Fundamentos da Programação de Computadores –  
Parte 2 – Páginas 93 a 144.



**RECORDANDO O  
WHILE**

# Recordando o While

- Aula passada: estrutura de repetição **while**
  - O que faz?



```
#include <iostream>
using namespace std;
int main(void) {
```

```
    int N;
```

```
    N = 3;
```

```
    while ( N <= 17 ) {
        cout << N << endl;
        N = N + 1;
    }
```

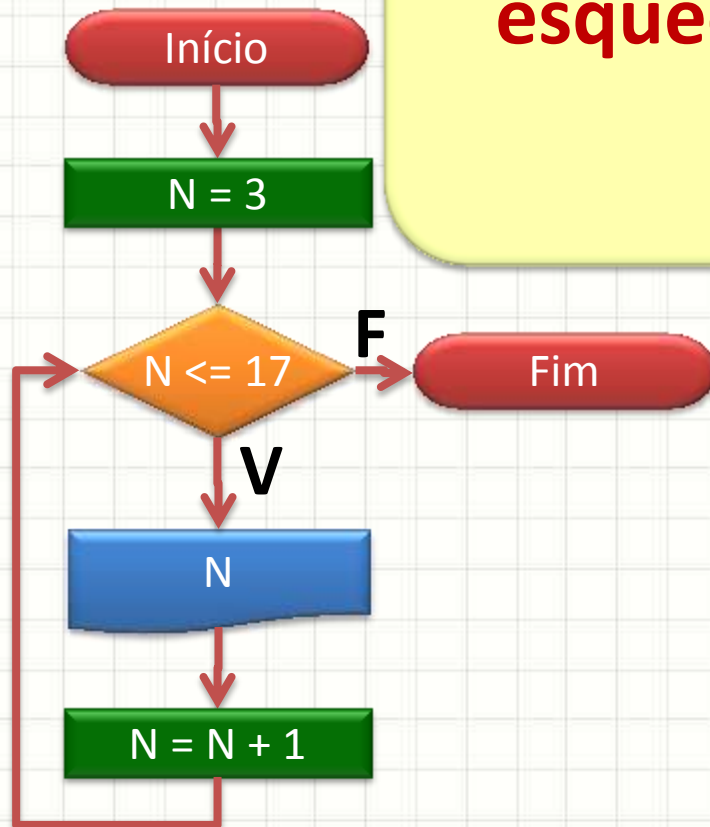
```
}
```



# Recordando o While

- Aula passada: estrutura de repetição **while**
  - O que faz

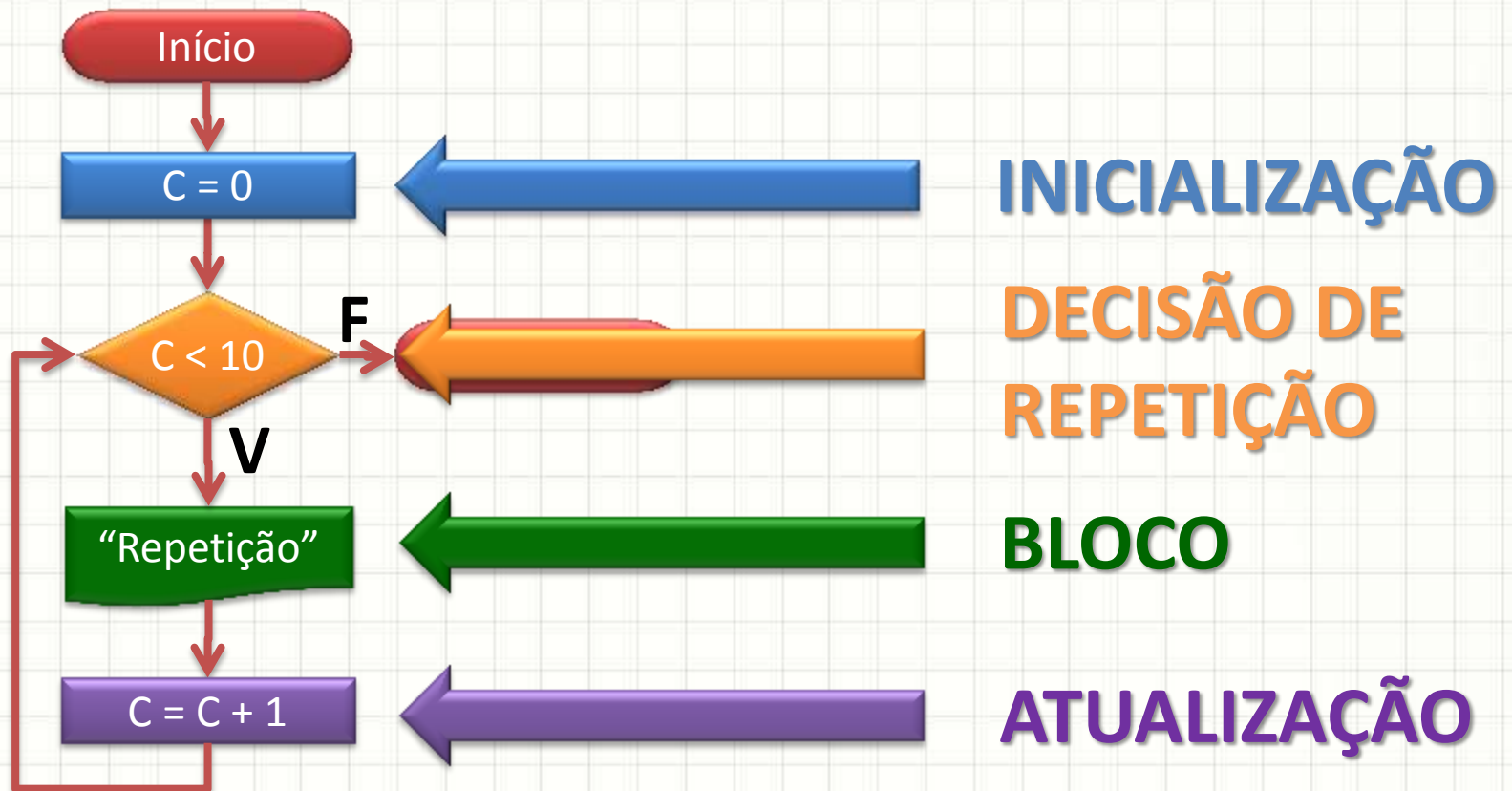
O que acontece se esquecermos essa linha?



```
ream>  
e std;  
{  
N = 3;  
while ( N <= 17 ) {  
    cout << N << endl;  
    N = N + 1;  
}
```

# Recordando o While

- Observe:
  - O que faz?



# Recordando o While

- No código...

```
#include <iostream>
using namespace std;
int main(void) {

    int CONT;

    CONT = 0;
    while ( CONT < 10 ) {
        cout << "Isso é uma repetição" << endl;
        CONT = CONT + 1;
    }
}
```

**INICIALIZAÇÃO**

**DECISÃO DE REPETIÇÃO**

**BLOCO**

**ATUALIZAÇÃO**



**Fácil esquecer  
um deles!**

le

```
#1
using namespace std;
int main(void) {

    int CONT;

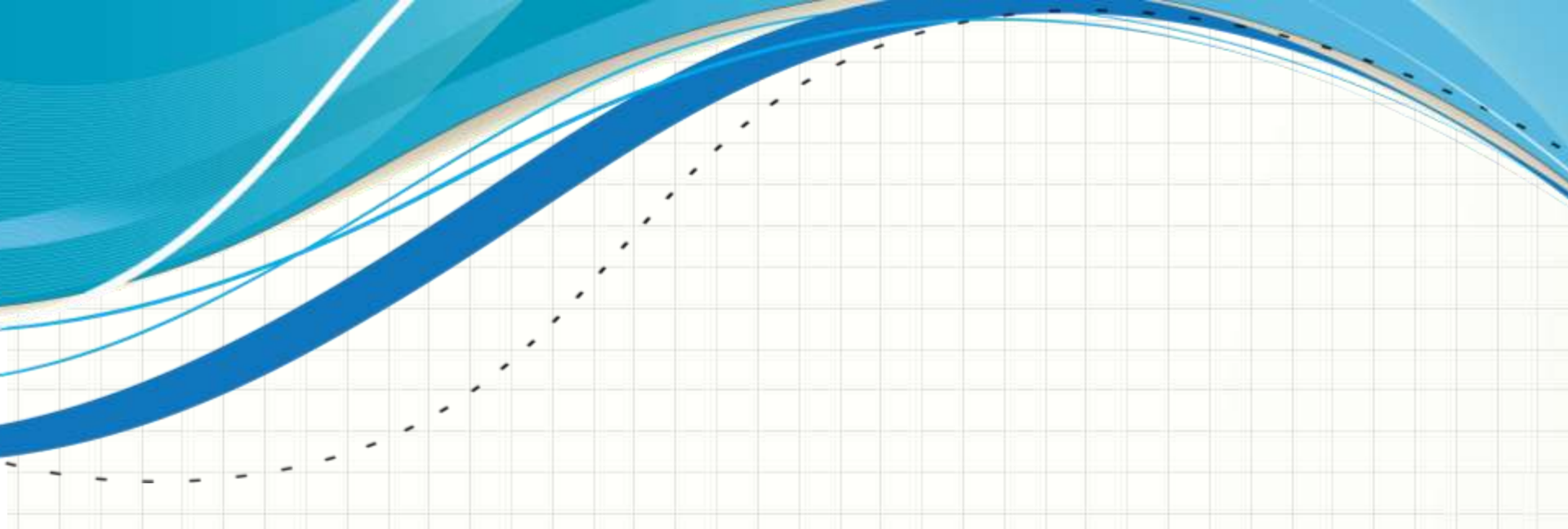
    CONT = 0;
    while ( CONT < 10 ) {
        cout << "Isso é uma repetição" << endl;
        CONT = CONT + 1;
    }
}
```

**INICIALIZAÇÃO**

**DECISÃO DE  
REPETIÇÃO**

**BLOCO**

**ATUALIZAÇÃO**



# A ESTRUTURA DE REPETIÇÃO FOR

# O que é a estrutura **for**

- Todos os elementos em uma única linha
  - Só o bloco fica “isolado”

```
#include <iostream>
using namespace std;
int main(void) {

    int CONT;

    CONT = 0;
    while ( CONT < 10 ) {
        cout << "Isso é uma repetição" << endl;
        CONT = CONT + 1;
    }
}
```

# O que é a estrutura **for**

- Todos os elementos em uma única linha
  - Só o bloco fica “isolado”

```
#include <iostream>
using namespace std;
int main(void) {

    int CONT;

    for (CONT = 0; CONT < 10; CONT = CONT + 1) {
        cout << "Isso é uma repetição" << endl;
    }
}
```

# O que é a estrutura for

- Todos os elementos em uma única linha
  - Só o bloco fica “isolado”

**INICIALIZAÇÃO**

**DECISÃO DE  
REPETIÇÃO**

**ATUALIZAÇÃO**

```
#include <iostream>
using namespace std;
int main(void) {
```

```
    int CONT;
```

```
    for (CONT = 0; CONT < 10; CONT = CONT + 1) {
        cout << "Isso é uma repetição" << endl;
    }
```

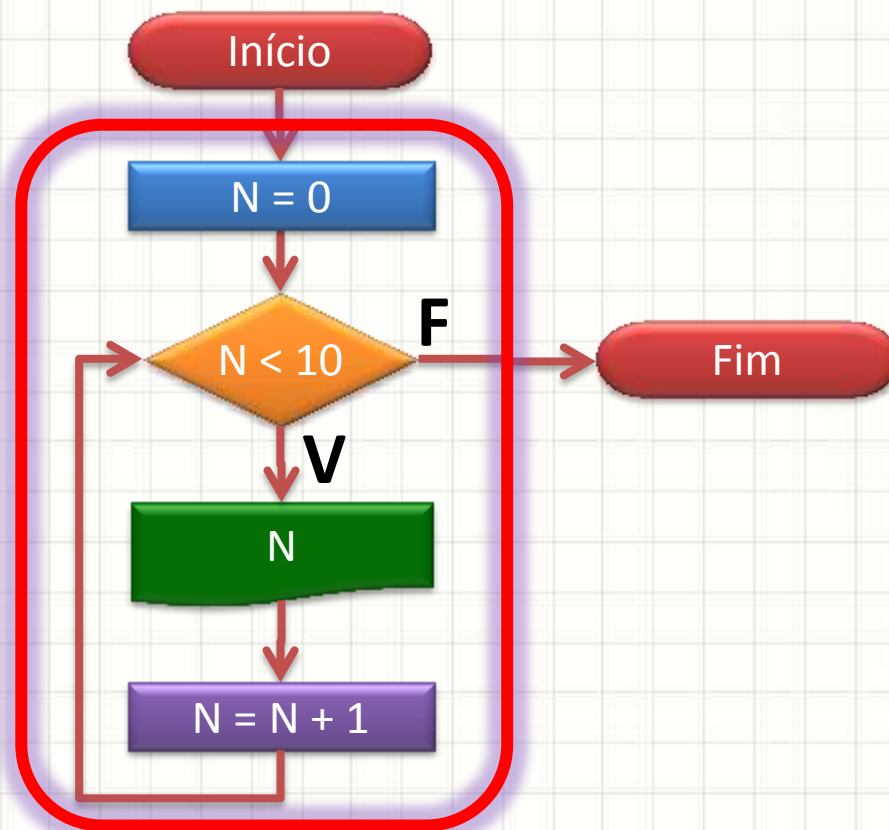
```
}
```

**BLOCO**



# Forma Geral do for

```
for ( inicialização; condição de repetição; atualização ) {  
    Executa enquanto a proposição for verdadeira  
}
```



# Leitura do for

```
for ( X = 0 ; X < 7 ; X = X + 2 ) {  
    cout << X << endl;  
}
```

Faça, a partir de  $X = 0$ , enquanto  $X < 7$  e contando de 2 em 2, a impressão de  $X$ .

# EXERCÍCIO

A) Faça um programa que apresente os números de 50 a 75.

# EXERCÍCIO

A) Faça um programa que apresente os números de 50 a 75.

```
#include <iostream>
using namespace std;
int main(void) {

    int C;

    for ( C = 50; C <= 75; C = C + 1) {
        cout << C << endl;
    }
}
```

# EXERCÍCIO

B) Modifique o programa anterior para que ele conte de 2 em 2.



# EXERCÍCIO

B) Modifique o programa anterior para que ele conte de 2 em 2.

```
#include <iostream>
using namespace std;
int main(void) {

    int C;

    for ( C = 50; C <= 75; C = C + 2) {
        cout << C << endl;
    }
}
```

# EXERCÍCIO

C) Modifique o programa para que imprima só números divisíveis por 5.

# EXERCÍCIO

C) Modifique para que imprima só números divisíveis por 5.

```
#include <iostream>
using namespace std;
int main(void) {

    int C;

    for ( C = 50; C <= 75; C = C + 2) {
        if ( C%5 == 0 ) {
            cout << C << endl;
        }
    }
}
```



# REPETIÇÃO COM DO~WHILE

# Repetição com Do~While

- Algumas vezes queremos que um procedimento seja executado “pelo menos uma vez”.
- Quando?
  - Esperar que um dado específico seja digitado...
    - É preciso ler a entrada **antes** de testá-la
  - Esperar um valor específico de um sensor
    - É preciso ler o sensor **antes** de testar o valor
  - Etc.



# Repetição com Do~While

- Observe: `#include <math.h>`  
`#include <iostream>`  
`using namespace std;`

```
int main(void) {  
    float N,R;  
    cout << "Digite um número positivo:";  
    cin >> N;  
    R = sqrt(N);  
    cout << "Raiz:" << R;  
}
```

E se o usuário digitar um número negativo?

Não seria legal poder repetir a pergunta?

# Repetição com Do~While

```
#include <math.h>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(void) {
```

```
    float N,R;
```

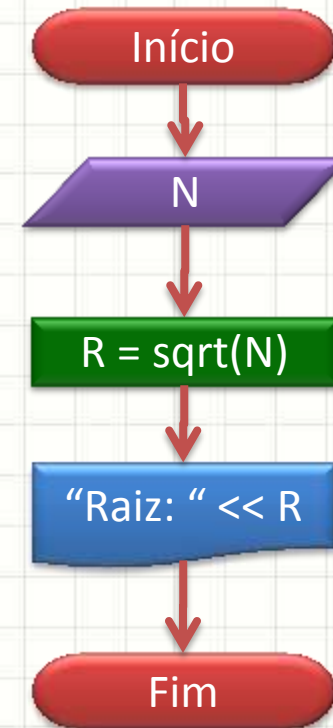
```
    cout << "Digite no. > 0: ";
```

```
    cin >> N;
```

```
    R = sqrt(N);
```

```
    cout << "Raiz: " << R;
```

```
}
```



# Repetição com Do~While

```
#include <math.h>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(void) {
```

```
    float N,R;
```

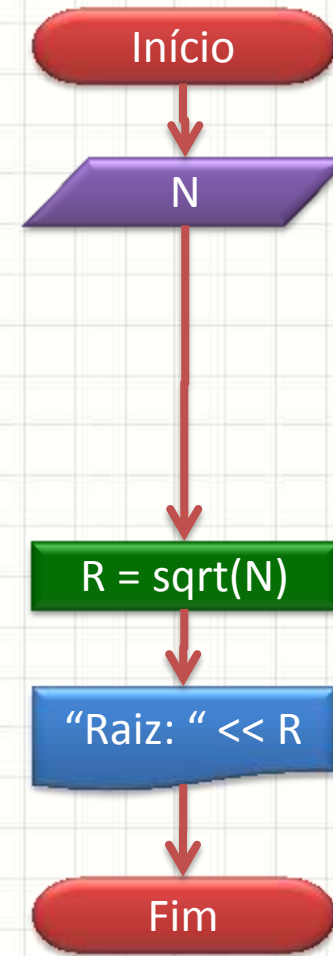
```
    cout << "Digite no. > 0: ";
```

```
    cin >> N;
```

```
    R = sqrt(N);
```

```
    cout << "Raiz: " << R;
```

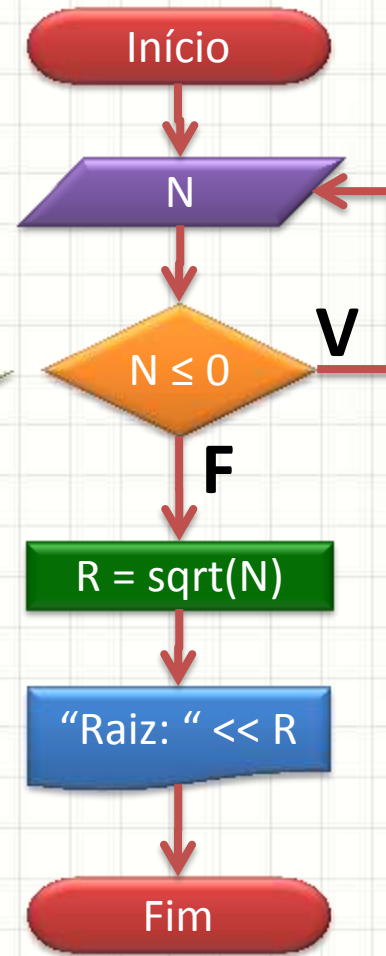
```
}
```



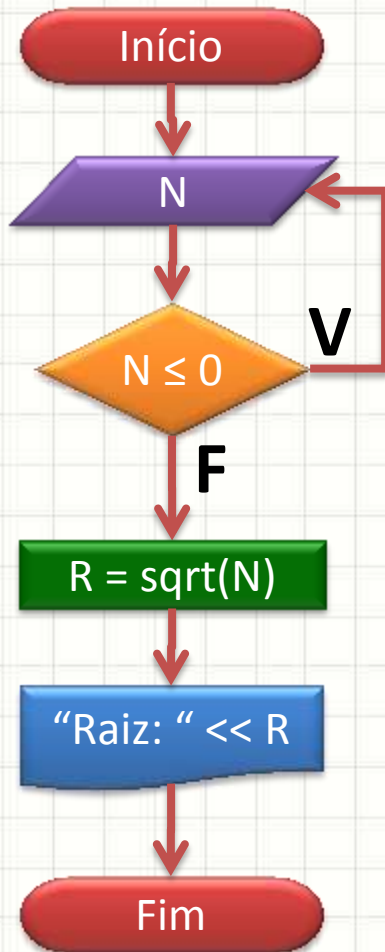
# Repetição com Do~While

```
#include <math.h>
#include <iostream>
using namespace std;
int main()
{
    float N;
    cout << "Digite um número: ";
    cin >> N;
    R = sqrt(N);
    cout << "Raiz: " << R;
}
```

**while  
serve?**



# Repetição com Do~While



```
#include <math.h>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(void) {
```

```
    float N,R;
```

```
    do {
```

```
        cout << "Digite no. > 0: ";
```

```
        cin >> N;
```

```
    } while ( N <= 0 );
```

```
    R = sqrt(N);
```

```
    cout << "Raiz: " << R;
```

```
}
```



# Forma Geral do **do~while**

**do {**

Executa enquanto a proposição for verdadeira

**} while ( condição de repetição );**

- Qual a diferença com relação ao **while** ?

**while ( condição de repetição ) {**

Executa enquanto a proposição for verdadeira

**}**

# EXERCÍCIO

A) Crie um menu para que ele contenha as seguintes opções:

- 1- Saldo
- 2- Extrato
- 3- Saque
- 4- Depósito

Para cada opção ele deve imprimir um texto que indique a opção selecionada... e ele não deve aceitar opções inválidas

# EXERCÍCIO

A) Crie um programa em C++ que apresente as seguintes opções:

1- Saldo

2- Extrato

3- Saque

4- Depósito

Para cada opção, o programa deve apresentar o texto que indica a operação e ele não deve executar nenhuma outra operação.

```
#include <iostream>
using namespace std;
int main(void) {
    int N;

    cout << "1- Saldo" << endl;
    cout << "2- Extrato" << endl;
    cout << "3- Saque" << endl;
    cout << "4- Depósito" << endl;
    do {
        cin >> N;
    } while (N < 1 || N > 4);

    if (N == 1) {
        cout << "Saldo!" << endl;
    }
    if (N == 2) {
        cout << "Extrato!" << endl;
    }
    if (N == 3) {
        cout << "Saque!" << endl;
    }
    if (N == 4) {
        cout << "Depósito!" << endl;
    }
}
```

# EXERCÍCIO

B.1) Analise os códigos e descubra qual é mais adequado para **while** e qual para **do~while**:

## Código 1

- a) Leia um número N;
- b)  $N = N * 2$ ;
- c) Se N for menor que 32, volta para o passo (b);
- d) Imprima N.

## Código 2

- a) Leia um número N;
- b) Enquanto N for menor que 32, repita (c)
- c)  $N = N * 2$ ;
- d) Imprima N.

# EXERCÍCIO

B.1) Analise os códigos e descubra qual é mais adequado para **while** e qual para **do~while**:

## Código 1

- a) Leia um número N;
- b)  $N = N * 2$ ;
- c) Se N for menor que 32, volta para o passo (b);
- d) Imprima N.



**do~while**

## Código 2

- a) Leia um número N;
- b) Enquanto N for menor que 32, repita (c)
- c)  $N = N * 2$ ;
- d) Imprima N.



**while**



# EXERCÍCIO

B.2) Implemente ambos os códigos:

## Código 1

- a) Leia um número N;
- b)  $N = N * 2$ ;
- c) Se N for menor que 32, volta para o passo (b);
- d) Imprima N.

## Código 2

- a) Leia um número N;
- b) Enquanto N for menor que 32, repita (c)
- c)  $N = N * 2$ ;
- d) Imprima N.

# EXERCÍCIO

## B.2) Implemente os códigos

### Código 1

- Leia um número N;
- $N = N * 2$ ;
- Se N for menor que 32, volta para o passo (b);
- Imprima N.

```
#include <iostream>
using namespace std;
int main(void) {

    float N;

    cout << "Digite um número: ";
    cin >> N;

    do {
        N = N * 2;
    } while (N < 32);

    cout << "Resultado: " << N << endl;
}
```

# EXERCÍCIO

B.2)

Implemente os códigos:

## Código 2

- a) Leia um número N;
- b) Enquanto N for menor que 32, repita (c)
- c)  $N = N * 2$ ;
- d) Imprima N.

```
#include <iostream>
using namespace std;
int main(void) {

    float N;

    cout << "Digite um número: ";
    cin >> N;

    while (N < 32) {
        N = N * 2;
    }

    cout << "Resultado: " << N << endl;
}
```

# EXERCÍCIO

B.3) Execute ambos os códigos para as entradas:

0

20

40

E responda: os resultados são sempre iguais? Por quê?

# EXERCÍCIO

B.3) Execute ambos os códigos para as entradas:

0

20

40

E responda: os resultados são sempre iguais? Por quê?

<u>Valor de Entrada:</u>	<u>Resultado Código 1</u>	<u>Resultado Código 2</u>
0	Travou	Travou
20	40	40
40	80	40





**CONCLUSÕES**

# Resumo

- Existem diversos tipos de estruturas de decisão
- Elas são intercambiáveis, isto é, tudo que se faz com um uma, é possível fazer com outra
- Dependendo da situação, cada uma delas é mais apropriada!
- **TAREFA!**
  - Lista de Exercícios 2!

# Próxima Aula



- Aprenderemos a criar códigos mais complexos...
  - Como reaproveitar?!
  - Modularização



**PERGUNTAS?**



**BOM DESCANSO  
A TODOS!**