Estruturas Condicionais

Op. Lógicos e Relacionais

INF110 – Programação I

Prof. Alcione/André Gustavo
DPI/UFV – 2020/1

UFV

Em aulas anteriores...

```
#include <iostream>
using namespace std;

int main() {

    int prova1, prova2, prova3, notafinal;

//Ler nota da prova 1, prova 2, prova 3
    cout << "Digite sua nota na prova 1: ";
    cin >> prova1;
    cout << "Digite sua nota na prova 2: ";
    cin >> prova2;
    cout << "Digite sua nota na prova 3: ";
    cin >> prova3;

//Calcular a nota final: somar as 3 notas
    notafinal = prova1 + prova2 + prova3 ;

//Escrever a nota final
    cout << "Sua nota final foi  ";
    cout << notafinal << endl ;

    return 0;
}
```

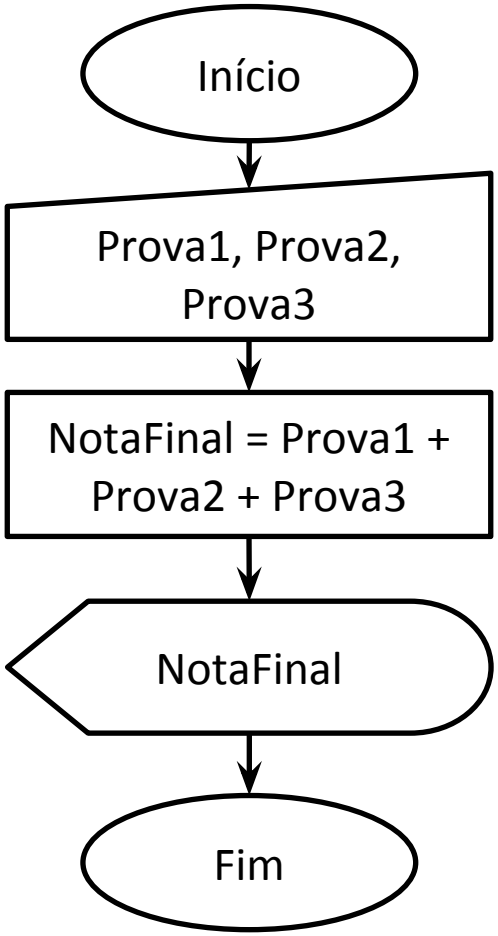
Completando o programa

- Precisamos agora informar se o estudante foi aprovado
- Para ser aprovado, deve ter nota final no mínimo 60
  - Ou seja, se a nota final for ≥ 60, foi aprovado

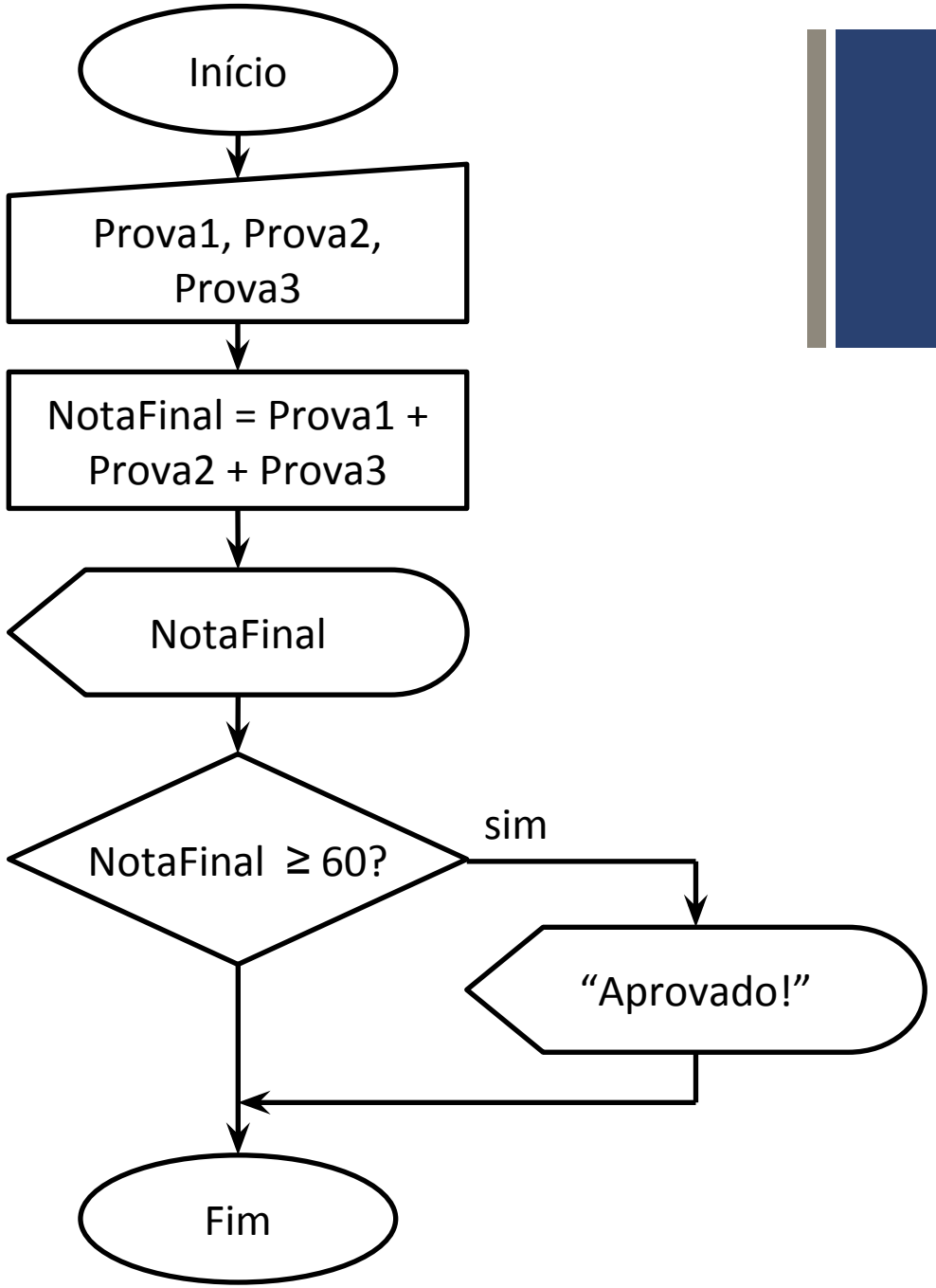
Pseudocódigo

- Ler Prova1, Prova 2, Prova3
- NotaFinal = Prova1 + Prova2 + Prova3
- Escrever NotaFinal
- **SE** NotaFinal >= 60
  - Escrever “Aprovado!”

Fluxograma


Início
Prova1, Prova2, Prova3
NotaFinal = Prova1 + Prova2 + Prova3
NotaFinal
Fim

Fluxograma


Início
Prova1, Prova2, Prova3
NotaFinal = Prova1 + Prova2 + Prova3
NotaFinal
NotaFinal ≥ 60?
sim
“Aprovado!”
Fim

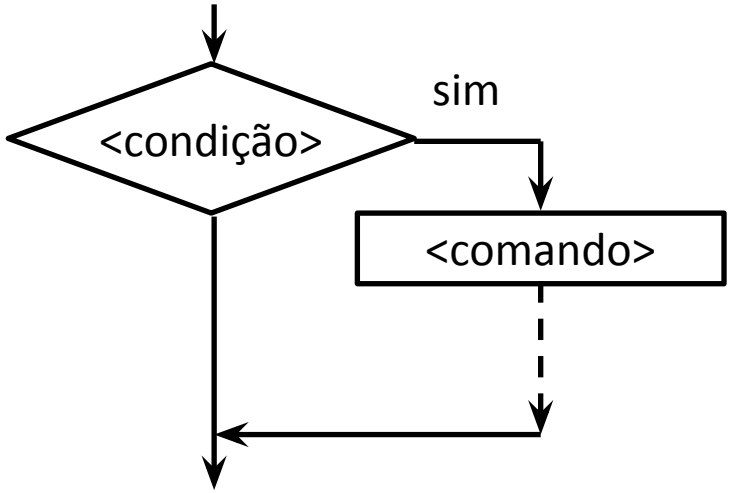
Código

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6      int prova1, prova2, prova3, notafinal;
7
8      //Ler notas da prova1, prova2, prova3
9      cout << "Digite sua nota na prova 1: ";
10     cin >> prova1;
11     cout << "Digite sua nota na prova 2: ";
12     cin >> prova2;
13     cout << "Digite sua nota na prova 3: ";
14     cin >> prova3;
15
16     //Calcular a nota final: somar as 3 notas
17     notafinal = prova1 + prova2 + prova3;
18
19     //Escrever a nota final
20     cout << "Sua nota final foi ";
21     cout << notafinal << endl;
22
23     if (notafinal >= 60)
24         cout << "Aprovado!" << endl;
25
26     return 0;
27 }
```

Estrutura condicional simples

- Permite que seja executado um comando (ou mais) se e somente se uma condição for satisfeita

```
SE  <condição>
   <comando>
   ...

FIM-SE

...
```




```
if (<condicao>)
{
     <comando>;
     ...
}
...
```

- A condição é geralmente uma expressão lógica e/ou relacional

Operadores relacionais

- Comparam duas expressões e retornam um valor lógico (V ou F)

Operador	Significado	
>	>	Maior que
<	<	Menor que
>=	≥	Maior que ou igual a
<=	≤	Menor que ou igual a
==	=	Igual a
!=	≠	Diferente de

Operadores relacionais

- CUIDADO!
  - O op. relacional para ≥ é >=, não use =>
  - O op. relacional para ≤ é <=, não use =<
  - O op. relacional para ≠ é `!=`, não use <>
  - O op. relacional para = é ==, não use =
    - O operador = é o operador de atribuição!
- Os demais geram erro de compilação, trocar == por = não!
  - É permitido, mas geralmente é um erro de lógica
  - `if (x=y)`
    - não verifica se x é igual a y
    - atribui o valor de y a x (e retorna o valor, dá verdadeiro se é ≠ 0)

Operadores – precedência

- Da maior para a menor precedência

Operador	Significado	Ordem
( )	Parênteses	De dentro pra fora, da esquerda para direita
* / %	Multiplicação, divisão, resto	Da esquerda para direita
+ -	Adição, Subtração	
=	Atribuição	Da direita para esquerda

Operadores – precedência

- Da maior para a menor precedência

Operador	Significado	Ordem
( )	Parênteses	De dentro pra fora, da esquerda para direita
* / %	Multiplicação, divisão, resto	Da esquerda para direita
+ -	Adição, Subtração	
< > <= >=	Relacionais <, >, ≤, ≥	
== !=	Relacionais = e ≠	
=	Atribuição	Da direita para esquerda

Exercícios

- Verdadeiro ou falso?
  - 5 >= 5
  - 3 - 2 <= 0
  - 5 – 2 * 2 >= 5
  - (3 – 2) != (5 - 4)
  - 9 / 2 == 4
- Qual o contrário de
  - A == B
  - A >= B

Exercícios

- Verdadeiro ou falso?
  - 5 >= 5 V
  - 3 - 2 <= 0 F
  - 5 – 2 * 2 >= 5 F
  - (3 – 2) != (5 - 4) F
  - 9 / 2 == 4 V
- Qual o contrário de
  - A == B A != B
  - A >= B A < B

Exercícios

1. Ler um valor inteiro e dizer se ele é par
2. Ler um ano e dizer se ele é bissexto
3. No programa das notas, informar se o aluno foi reprovado (nota < 60)
4. No programa anterior, informar a situação do aluno:
   aprovado (se nota ≥ 60) ou reprovado (se nota < 60)

Estrutura condicional composta

- Estruturas condicionais simples permitem que seja executado um comando (ou mais) se e somente se uma condição for satisfeita.
- E se quisermos executar algo diferente se a condição for falsa?
- Temos a *Estrutura condicional composta*

Completando o programa (II)

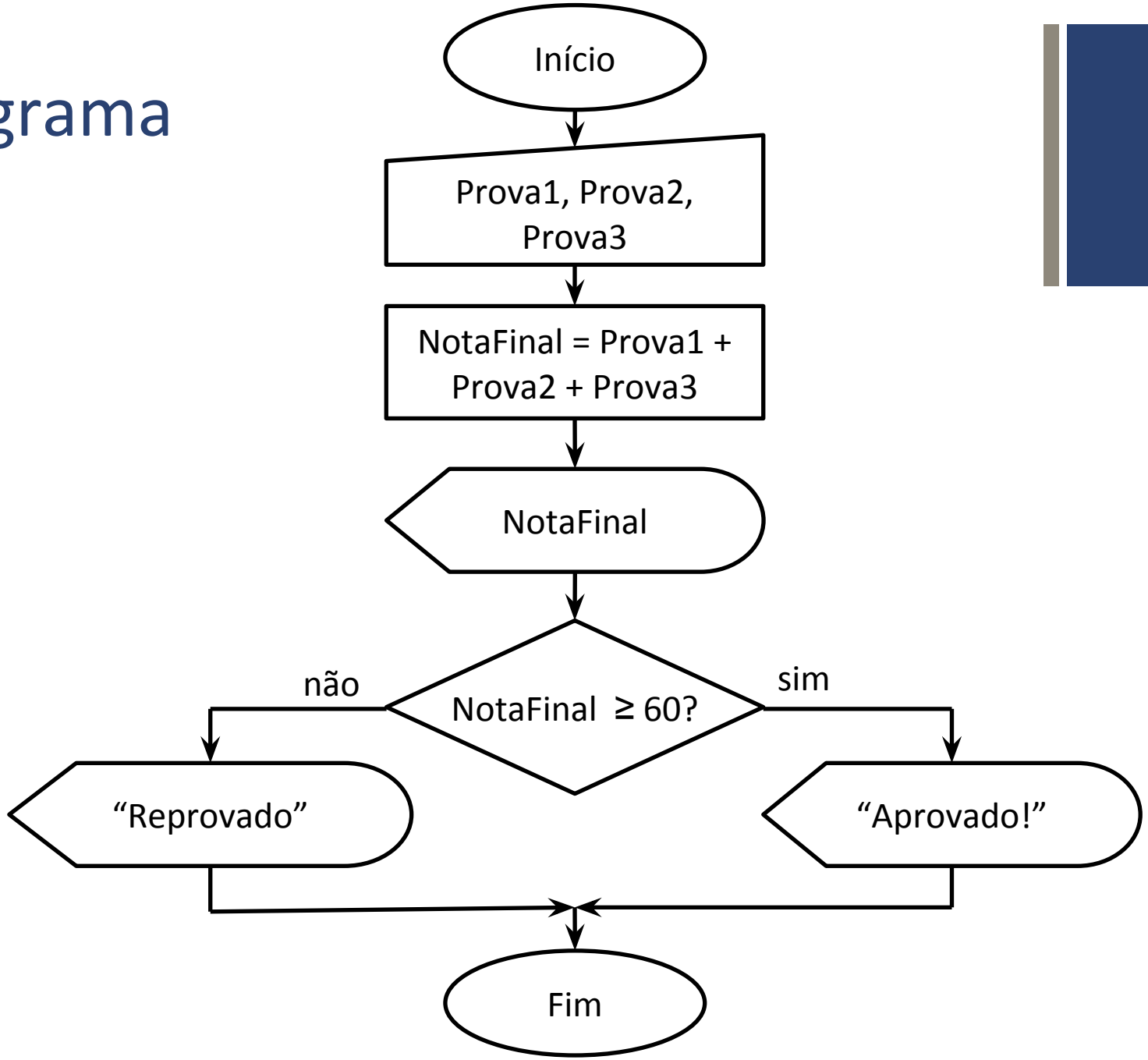
- Precisamos agora dizer se o estudante foi aprovado ou não
- Para ser aprovado, deve ter nota final no mínimo 60
  - Ou seja, se a nota final for ≥ 60, foi aprovado
  - Se a nota for < 60, foi reprovado

Pseudocódigo

- Ler Prova1, Prova 2, Prova3
- NotaFinal = Prova1 + Prova2 + Prova3
- Escrever NotaFinal
- **SE** NotaFinal >= 60
  - Escrever "Aprovado!"

  SENÃO
  - Escrever "Reprovado."

Fluxograma


Início
Prova1, Prova2, Prova3
NotaFinal = Prova1 + Prova2 + Prova3
NotaFinal
NotaFinal ≥ 60?
não
sim
“Reprovado”
“Aprovado!”
Fim

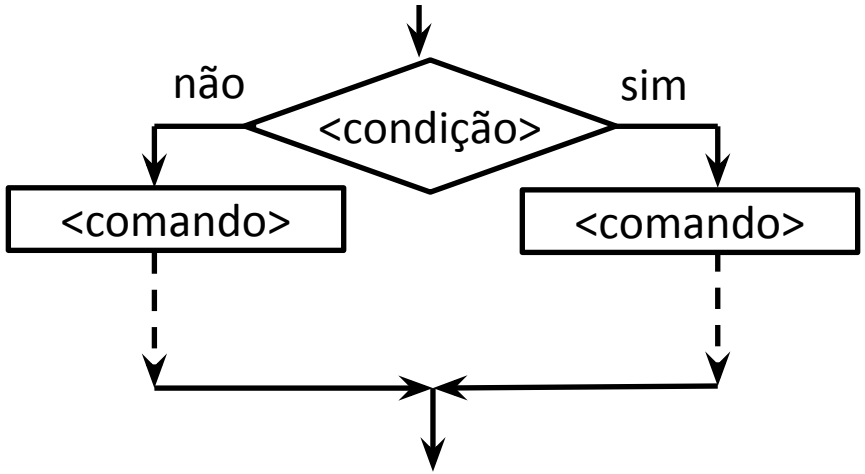
Código

```
#include <iostream>
using namespace std;

int main() {

    int prova1, prova2, prova3, notafinal;

    //Ler notas da prova1, prova2, prova3
    cout << "Digite sua nota na prova 1: ";
    cin >> prova1;
    cout << "Digite sua nota na prova 2: ";
    cin >> prova2;
    cout << "Digite sua nota na prova 3: ";
    cin >> prova3;

    //Calcular a nota final: somar as 3 notas
    notafinal = prova1 + prova2 + prova3;

    //Escrever a nota final
    cout << "Sua nota final foi ";
    cout << notafinal << endl;

    if (notafinal >= 60)
        cout << "Aprovado!" << endl;
    else
        cout << "Reprovado." << endl;

    return 0;
}
```

Estrutura condicional composta

- Permite que se decida executar uma sequência de comandos ou outra, dependendo da condição ser satisfeita ou não.

```
SE  <condição>
    <comando>
    ...
SENÃO
    <comando>
    ...
FIM-SE
...
```




```
if (<condicao>)
{
    <comando>;
    ...
}
else
{
    <comando>;
    ...
}
```

- Note que será executada uma, e somente uma das sequências

Observações

- Cada **if** pode ter no máximo um **else**
- Todo **else** precisa ter um **if** (é associado ao if anterior)
- É obrigatório usar ( ) na condição
- Não se coloca ; após a condição (se colocar, termina o if!)

Exercícios

5. Ler um valor inteiro e dizer se ele é par ou ímpar

6. Ler o consumo (comida, bebida, sobremesa) e o valor pago, e informar o troco somente se o valor pago for suficiente. Se não for, imprimir uma mensagem.

7. Certa ponte permite veículos de até 3m de largura e 10 ton. Ler a largura e peso de um veículo e dizer se ele pode passar na ponte.

Operadores lógicos

- Operadores relacionais permitem comparar dois valores
- E se precisarmos comparar vários?
- Existem os operadores lógicos, que combinam resultados de operadores relacionais e lógicos

Operadores lógicos

- Retornam um valor lógico (V ou F)

Operador	Significado	
&&	E	V apenas de os dois (um E outro) forem V
||	OU	V se um dos dois (um OU outro) for V ou seja, F apenas se os dois forem F
!	NÃO	Negação (V se F, F se V)

Código

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6      int largura, peso;
7
8      cout << "Digite a largura e peso do veiculo: ";
9      cin >> largura >> peso;
10
11     //Testa se pode passar
12     if (largura <= 3 && peso <= 10)
13         cout << "Pode passar na ponte\n";
14     else
15         cout << "Nao pode passar na ponte\n";
16
17     return 0;
18 }
```

Álgebra booleana (tabela verdade)

X	Y	X && Y	X || Y
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

X	! X
V	F
F	V

Operadores – precedência

- Da maior para a menor precedência

Operador	Significado	Ordem
( )	Parênteses	De dentro pra fora, da esquerda para direita
!	Lógico NÃO	Da direita para esquerda
* / %	Multiplicação, divisão, resto	Da esquerda para direita
+ –	Adição, Subtração	
< > <= >=	Relacionais <, >, ≤, ≥	
== !=	Relacionais = e ≠	
&&	Lógico E	
||	Lógico OU	
=	Atribuição	Da direita para esquerda

Exercícios

- Verdadeiro ou falso?
  - 5 > 4 && 10 < 20
  - 5 > 4 && 10 > 20
  - 5 > 4 || 10 < 20
  - 5 < 4 && 10 > 20
  - ! (5 < 4) && 10 < 20
  - ! (5 < 4 && 10 < 20)
  - ! (5 < 4) && 10 > 20
  - ! (5 < 4 && 10 > 20)
- Qual o contrário de
  - A > B && C <= D

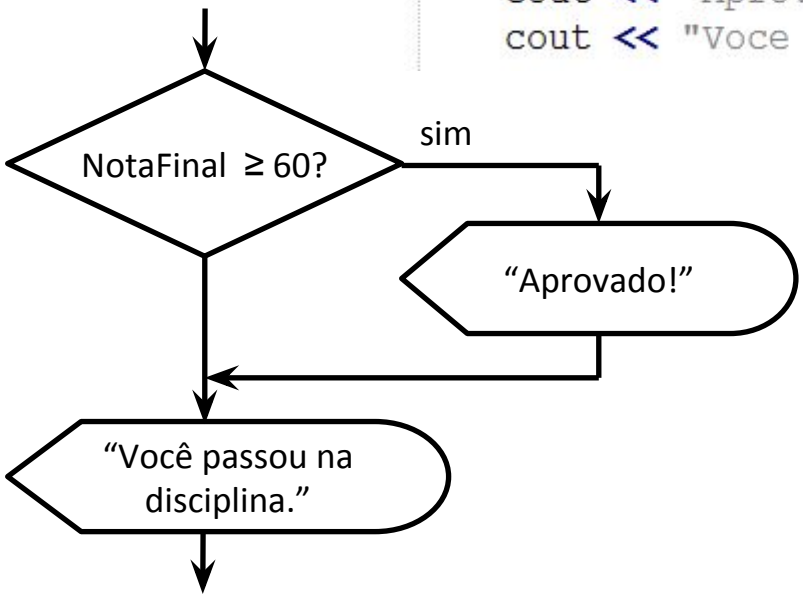
Exercícios

- Verdadeiro ou falso?
  - 5 > 4 && 10 < 20 V
  - 5 > 4 && 10 > 20 F
  - 5 > 4 || 10 < 20 V
  - 5 < 4 && 10 > 20 F
  - ! (5 < 4) && 10 < 20 V
  - ! (5 < 4 && 10 < 20) V
  - ! (5 < 4) && 10 > 20 F
  - ! (5 < 4 && 10 > 20) V
- Qual o contrário de
  - A > B && C <= D A <= B || C > D

Bloco de comandos

- Em C++, blocos de comandos são delimitados por { }
- Sem eles, os comandos condicionais executariam apenas um comando, o próximo
- Exemplo

```
if (notafinal >= 60)                    //ERRADO!!
    cout << "Aprovado!" << endl;
    cout << "Voce passou na disciplina." << endl;
```

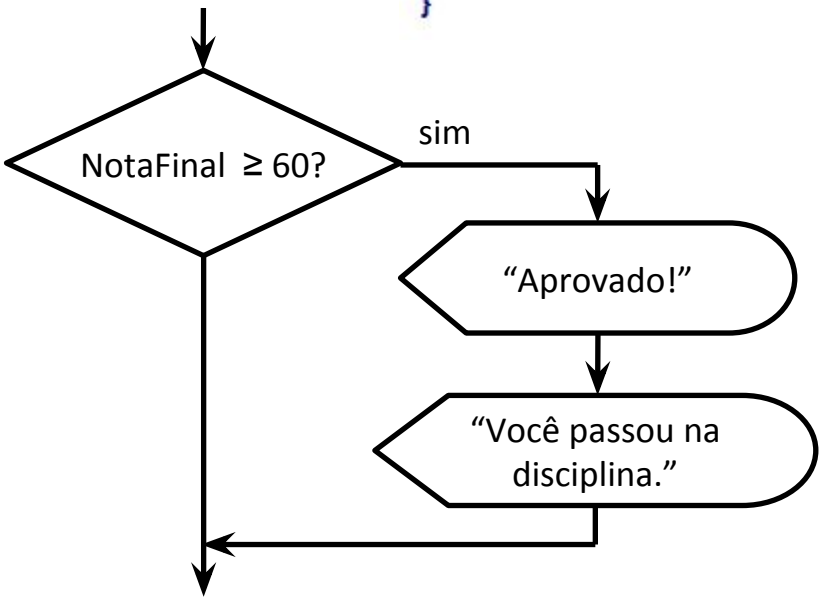


Note o erro no fluxograma

Bloco de comandos

- Com eles, os comandos condicionais executam o bloco
- Exemplo

```
if (notafinal >= 60)
{
    cout << "Aprovado!" << endl;
    cout << "Voce passou na disciplina." << endl;
}
```



Note o resultado no fluxograma

There are two types of people.

Programmers will know.

Indentação

- **Indentação** (neologismo do inglês *indentation*) é um **recuo** aplicado no código fonte para ressaltar a estrutura do algoritmo
- Na maioria das linguagens, a indentação tem por objetivo ressaltar a estrutura do algoritmo, aumentando a legibilidade
- Em algumas linguagens, entretanto, é obrigatória (p.ex. Python)

Indentação – comand. condicionais

- A sequência de comandos do **if** e do **else** deve ser indentada
- Isso é feito com TAB ou espaços (geralmente 3)
- Alguns editores fazem indentação automaticamente

- A indentação aumenta a legibilidade
- Mas não modifica nada pro compilador C++!
  - Uma sequência de comandos indentada não é um bloco
  - A não ser quando delimitada por { }'s

Aimpo rtân ciadaInd entaç ão

- Comoseriaesseslideparavocêlerseeunãousasseespaçoentreasletras?
- Es eeuus asseesp açomasn oloc alqu eeuqu ises se?
- Seráqu eumaau latodac omsli desass ims eriapr odutiv a?

- Percebem como fica quase ilegível?
- É preciso muito esforço para ler uma simples frase
- O mesmo ocorre quando analisamos código sem indentação!

Citações

- Martin Flower

  “Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”

- Donald E. Knuth

  “Programming is best regarded as the process of creating works of literature, which are meant to be read.”