



Capítulo 3. Estruturas Condicionais

Tópico 1. If ... Else

Explicação 1. Às vezes é necessário usar uma estrutura condicional. O “if”, que quer dizer “se” em português, verifica *se* uma **condição** é verdadeira. Se for, faz o que foi especificado no bloco. Opcionalmente, podemos usar o “else” para dizer o que o programa fará caso a condição seja falsa.

Exemplo 1. Como funciona o if:

```
if (condição)
{
    código a ser executado se a condição for verdadeira
}
else
{
    código a ser executado se a condição for falsa
}
```

Exemplo 2. Verificando se um número é maior que 20.

```
int x;
cin >> x;

if (x > 20)
{
    cout << "O número digitado é maior que 20;
}
```

Neste exemplo, se um número for maior do que 20, a mensagem “O número digitado é maior que 20” será exibida na tela. Caso contrário, nada acontecerá.

Exemplo 3. Verificando se um número é menor que 50.

```

int x;
cin >> x;

if (x < 50)
{
    cout << "O número é menor que 50";
}
else
{
    cout << "O número NÃO é menor que 50";
}

```

Neste exemplo, se x for menor que 50, a mensagem "O número é menor que 50" será exibida na tela. **Caso contrário**, a mensagem "O número NÃO é menor que 50" será exibida na tela. Observe bem o papel do "else" nessa estrutura.

Explicação 2. Uma *condição* deve ser digitada dentro dos parênteses do if. A condição sempre retorna um valor booleano, que pode ser verdadeiro ou falso. Se a condição for verdadeira, o bloco do if será executado. Se a condição for falsa, então o bloco do else será executado. Se não houver "else", nada acontece. Uma condição, portanto, é o conteúdo dos parênteses do if. Os operador que usamos nas condições, como por exemplo, "maior que", "menor que", "igual a", nós chamamos de operadores condicionais. Eis alguns operadores condicionais, e o que eles fazem.

Operador	Significado
==	Igual. Observe que são dois sinais de igual, e não um. x == y
!=	Diferente x != y
>	Maior x > y
<	Menor x < y
>=	Maior ou igual x >= y
<=	Menor ou igual x <= y

Exemplo 1. Usando o operador igual.

```

if ( x == y )
{
    cout << "x é igual a y";
}

```

Exemplo 2. Usando o operador diferente.

```
if ( x != y )
{
    cout << "x é diferente de y";
}
else
{
    cout << "x não é diferente de y; ou seja, x é igual a y";
}
```

Exemplo 3. Usando os operadores de maior e menor.

```
if ( x < y )
{
    cout << "x é menor do que y";
}

if ( x > y )
{
    cout << "x é maior que y";
}
else
{
    cout << "x não é maior que y; ou seja, x é menor que y";
}
```

Exemplo 4. Usando os operadores de maior ou igual e menor ou igual.

```
if ( x <= y )
{
    cout << "x é menor OU igual a y";
}
else
{
    cout << "x NÃO é [maior ou igual] a y; ou seja, x é menor que y";
}
```

Explicação 3. Você pode escrever várias condições dentro de um único if, usando os operadores “and” ou “&&” significando “e” (ou seja, verifica se as duas condições são verdadeiras); e também usando os operadores “or” ou “||”, significando “ou” (para ver se pelo menos uma das condições é verdadeira).

Operador	Significado
and &&	Significa “e”. x == y and y == 2 Também pode ser escrito assim: x == y && y == 2
or 	Significa “ou”. x != y or y == 4 Também pode ser escrito assim: x != y y == 4

Exemplo 1. Neste exemplo, queremos saber se a pessoa é homem, e se tem 65 anos.

```
int sexo, idade;
cout << "Você é homem? Sim é 1; Não é 2.";
cin >> sexo;
cout << "Qual a sua idade?"
cin >> idade;
if (sexo == 1 && idade == 65)
{
    cout << "Parabéns! Ganhou um check-up grátis!";
}
else
{
    cout << "Você não preenche os requisitos.";
}
```

A verdade é que o “Parabéns! Ganhou um check-up grátis!” só será mostrado na tela se, e somente se, a pessoa for homem *e também* tiver 65 anos. Se a pessoa for homem mas não tiver 65 anos, ou se tiver 65 anos, não for homem, ou ainda, se não preencher a nenhuma das condições, o bloco do “else” será executado: “Você não preenche aos requisitos”.

Exemplo 2. Para conseguir entrar no brinquedo, a criança deve ter 12 anos ou mais.

Mas, se não tiver 12 anos, a gente deixa passar se ele aparentar ter 12 anos. Ou seja, a condição será verdadeira se a criança tiver 12 anos **ou** aparente ter 12 anos.

```
int idade, aparenta;
cout << "Qual a sua idade?";
cin >> idade;
cout << "Você aparenta ter 12 anos? 1 é sim, 2 é não.";
cin >> aparenta;
if ( idade >= 12 || aparenta == 1 )
{
    cout << "Pode passar!";
}
else
{
    cout << "Ah... que peninha!!!!";
}
```

Vejam, agora, as possíveis situações em que este programa pode ser usado.

```
Qual a sua idade? 10
Você aparenta ter 12 anos? 1 é sim, 2 é não. 1
Pode passar!
```

```
Qual a sua idade? 12
Você aparenta ter 12 anos? 1 é sim, 2 é não. 2
Pode passar!
```

```
Qual a sua idade? 12
Você aparenta ter 12 anos? 1 é sim, 2 é não. 1
Pode passar!
```

```
Qual a sua idade? 10
Você aparenta ter 12 anos? 1 é sim, 2 é não. 2
Ah... que peninha!!!!
```

Explicação 4. Nem sempre o oposto do verdadeiro é falso. Ou melhor... podemos ter vários falsos para um único verdadeiro. Você deve ter observado que quando temos várias condições dentro de um if, podemos ter vários falsos. Por exemplo, se alguém diz “vou à praia se fizer sol *e se* acordar cedo”, temos um único verdadeiro, e três falsos. O verdadeiro é: “faz sol e também acordei cedo; logo vou à praia”. Os três falsos são: (1) “faz sol, mas não acordei cedo”, (2) “acordei cedo, mas não faz sol” e (3) “nem faz sol, nem acordei cedo”. Logo, “não vou à praia”.

Exemplo 1. Quando usamos E, estamos sendo mais específicos, mas restritos. Como no exemplo da praia: “só vou à praia se fizer sol e acordar cedo”.

```
cout << "Está fazendo sol? SIM é 1; NÃO é 2. ";
int sol;
cin >> sol;

cout << "Você acordou cedo? SIM é 1; NÃO é 2. ";
int cedo;
cin >> cedo;

if ( sol == 1 && cedo == 2 )
{
    cout << "Eba! Você vai a praia!!" << endl;
}
else
{
    cout << "Ih. Se deu mau!" << endl;
}
```

Agora, vamos simular o comportamento do programa.

```
Está fazendo sol? SIM é 1; NÃO é 2. 1
Você acordou cedo? SIM é 1; NÃO é 2. 2
Ih. Se deu mau!
```

```
Está fazendo sol? SIM é 1; NÃO é 2. 2
Você acordou cedo? SIM é 1; NÃO é 2. 1
Ih. Se deu mau!
```

```
Está fazendo sol? SIM é 1; NÃO é 2. 2
Você acordou cedo? SIM é 1; NÃO é 2. 2
Ih. Se deu mau!
```

```
Está fazendo sol? SIM é 1; NÃO é 2. 1
Você acordou cedo? SIM é 1; NÃO é 2. 1
Eba! Você vai a praia!!
```

Em suma: com o E, temos 25% de chances de que a condição seja verdadeira.

Exemplo 2. Já quando usamos o OU, estamos sendo mais liberais. Se uma das condições for verdadeira, então o bloco do if é executado. O else será executado apenas se todas as condições forem falsas. Neste exemplo, eu quero ir de qualquer jeito à praia. Se estiver sol OU se alguém me chamar.

```
cout << "Está sol? SIM é 1; NÃO é 2. ";
int sol;
cin >> sol;

cout << "Alguém chamou você? SIM é 1; NÃO é 2. ";
int chamou;
cin >> chamou;

if ( sol == 1 || chamou == 1 )
{
    cout << "Eba!" << endl;
}
else
{
    cout << "Ó vida, ó céus, ó azar!" << endl;
}
```

Agora, simulando:

```
Está sol? SIM é 1; NÃO é 2.          1
Alguém chamou você? SIM é 1; NÃO é 2. 2
Eba!

Está sol? SIM é 1; NÃO é 2.          2
Alguém chamou você? SIM é 1; NÃO é 2. 1
Eba!

Está sol? SIM é 1; NÃO é 2.          1
Alguém chamou você? SIM é 1; NÃO é 2. 1
Eba!

Está sol? SIM é 1; NÃO é 2.          2
Alguém chamou você? SIM é 1; NÃO é 2. 2
Ó vida, ó céus, ó azar!
```

Explicação 5. O tipo booleano. Na grande maioria das vezes, trabalhamos usando tipos numéricos, como **int** e **float** por exemplo. Mas há um tipo especial, o **bool** que armazena dois estados: verdadeiro (**true**) e falso (**false**). Esse tipo de variável pode ser usado no if.

Exemplo 1. Declaração e uso de variáveis do tipo bool.

```
bool x;
x = true;
bool y = false;
```

No caso, a variável x está como **true**, isto é, armazenando o valor verdadeiro; e a variável y está como **false**.

Explicação 6. Se a gente usar uma variável **bool** dentro do **if**, o resultado é bem interessante. Você talvez não veja utilidade alguma nisso, mas é importante compreender o que está acontecendo. Lembre-se que o bloco do **if** é executado sempre que a condição é verdadeira (**true**). Se a condição for falsa (**false**), o bloco do **else** é executado. Assim sendo, se em vez de condições, a gente colocar o **true** ou o **false** diretamente no **if**, isso é entendido por si só como o resultado de alguma condição.

Exemplo 1. True no if.

```
if (true)
{
    cout << "A condição é verdadeira" << endl;
}
else
{
    cout << "A condição é falsa" << endl;
}
```

Neste exemplo, será mostrada a frase “a condição é verdadeira.

Exemplo 2. False no if.

```
if (false)
{
    cout << "A condição é verdadeira" << endl;
}
else
{
    cout << "A condição é falsa" << endl;
}
```

Exemplo 3. Uma variável booleana como true, no if.

```
bool x = true;
if (x)
{
    cout << "x é verdadeiro" << endl;
}
else
{
    cout << "x é falso" << endl;
}
```

Exemplo 4. Uma variável booleana como false, no if.

```
bool y = false;
if (y)
{
    cout << "y é verdadeiro" << endl;
}
else
{
    cout << "y é falso" << endl;
}
```

Em resumo, não confie no y. Ele é falso.

Explicação 7. É muito importante compreender as variáveis booleanas, porque algumas funções retornam valores booleanos, e precisaremos trabalhar com eles. Estudaremos funções que retornam valores posteriormente, neste curso.

Tópico 2. Switch ... Case

Explicação 8. A palavra switch prende uma variável na memória, para ser analisada. Quem faz a análise é o "case". Poderíamos fazer a mesma coisa com um monte de if's, uns dentro dos outros, mas é melhor fazer com o case. É para isso que ele existe.

Exemplo 1. Verifique a opção que o usuário vai digitar.

```
cout << "Digite 2, 4, 6 ou 8: ";
int o;
cin >> o;

switch (o)
{
case 2:
    {
        cout << "Feijão com arroz" << endl;
        break;
    }
case 4:
    {
        cout << "Feijão no prato" << endl;
        break;
    }
case 6:
    {
        cout << "Quem é o freguês?" << endl;
        break;
    }
case 8:
    {
        cout << "Arroz com biscoito!" << endl;
        break;
    }
}
```

Explicação 9. A estrutura do case é um pouco complexa. Mas não ininteligível. Primeiro, o **switch** seleciona a variável que ficará “presa” para análise. Depois, você deve abrir um bloco, usando as já conhecidas chaves. Dentro desse bloco, você escreve **case** seguido do número a ser verificado. Depois do número, usa-se **dois pontos**. Isso mesmo. Uma coisa inédita até agora, que o C++ herdou da linguagem C. E a seguir, inicia-se um novo bloco. Assim, temos que o próprio “switch” é um bloco, que contém diversos blocos de case, quantos forem necessários.

Exemplo 1. Vamos selecionar uma variável para análise. Ela vai ficar presa na memória.

```
int x;
cout << "Digite um número: ";
cin >> x;

switch (x)
{
}
```

Este exemplo está incompleto, pois não fazemos quase nada com a variável x; apenas “prendemos” a mesma na memória. Observe que switch inicia um bloco. O próximo exemplo é a continuação deste.

Exemplo 2. Neste exemplo, dentro do switch, estamos escrevendo novos blocos para analisar a variável que está presa.

```
switch (x)    //Aqui prendemos a variável
{
  case 8:
  {
    cout << "X é igual a oito" << endl;
    break;
  }

  case 80:
  {
    cout << "X é igual a oitenta" << endl;
    break;
  }
}
```

Observe que, dentro de cada bloco do “case”, temos a palavra “break” na segunda linha. É importante usar “break”. Significa, por exemplo, que se x for igual a 8, somente o bloco do “case 8” será executado. No fim, o “break” vai sair do switch inteiro. Se você não usar “break”, todos os outros cases serão executados. E a gente não quer isso.

Explicação 10. Assim como o “if”, o “switch” tem o seu próprio “else”. Na verdade, usamos a palavra “default” (isto é, *padrão*) para que o switch faça alguma coisa, se todas as verificações foram falsas.

Exemplo 1. Verificar se o usuário digitou 1, 2 ou 3. Se digitou qualquer outra coisa... bem... ele deveria ser mais atento ao que está escrito na tela!

```
cout << "Digite 1, 2 ou 3: ";
int d;
cin >> d;

switch (d)
{
    case 1:
    {
        cout << "A primeira opção foi selecionada" << endl;
        break;
    }

    case 2:
    {
        cout << "A segunda opção foi selecionada" << endl;
        break;
    }

    case 3:
    {
        cout << "A terceira opção foi selecionada" << endl;
        break;
    }

    default:
    {
        cout << "Poxa!" << endl;
        break;
    }
}
```

No exemplo, o bloco do “default” só será executado se todas as verificações falharem. Ou seja, se o usuário não sabe ler.

Explicação 11. A estrutura “switch” é muito útil quando temos um programa que apresenta várias opções, e de acordo com a opção escolhida, o programa é direcionado para uma função (void) do programa. Estudemos funções em um momento posterior neste curso.