



Capítulo 1. Visão geral do C++

Tópico 1. Programando em C++ sem usar Orientação a Objetos

Explicação 1. C++ é uma linguagem orientada a objetos, assim como o Smalltalk e o Java. Entretanto, a programação orientada a objetos não será vista no início deste curso. Isso porque orientação a objetos é um conceito um pouco difícil, que vai exigir que você já tenha uma compreensão básica de programação em C++. Isso significa que você vai, pelo menos inicialmente, programar em C++ sem usar todos os recursos dessa linguagem. O C++ é uma linguagem derivada do C. Por isso, as duas usam a mesma forma de escrever, os mesmos sinais, e as mesmas regras. A linguagem C não é orientada a objetos. De qualquer modo, as diferenças entre as duas linguagens começam a ficar claras. Abaixo, veja um exemplo em C e C++, respectivamente, para exibir uma frase na tela. No primeiro exemplo abaixo, em C, a mensagem é exibida na tela através de uma função. No segundo exemplo, em C++, a mensagem é direcionada para uma palavra `cout`. Isso significa que não é absolutamente correto a gente dizer que “programação em C++ sem usar orientação a objetos é o mesmo que programar em C”. As linguagens são, evidentemente, muito parecidas. Mas não são iguais.

Exemplo 1. Escrever uma frase na tela usando C.

```
printf("Uma frase na tela\n");
```

Exemplo 2. Escrever uma frase na tela usando C++.

```
cout << "Uma frase na tela" << endl;
```

Explicação 2. C++ é um superconjunto de C. O que você faz em C, pode fazer em C++ também. C++ é mais abrangente, e de certo modo, mais fácil (isso se a gente usá-la sem orientação a objetos: ela é mais fácil em relação ao C). Além disso, com o suporte a orientação a objetos, que estudaremos mais tarde neste curso, podemos fazer mais coisas de maneira mais ordenada.

Tópico 2. Fim de Linha

Explicação 3. Em C++, toda linha que não inicie um bloco (falaremos de blocos em breve) devem ser finalizadas com um sinal de ponto-e-vírgula (;). Isso significa que se você começar a digitar código, ficar teclando <ENTER> e não pôr ponto-e-vírgula, o compilador (que é um programa que transforma o seu código em uma aplicação) irá entender tudo aquilo como se fosse uma única linha em C++.

Exemplo 1. A forma correta de escrever linhas em C++.

```
linha um;  
linha dois;  
linha três;
```

Exemplo 2. A forma **errada** de escrever linhas em C++.

```
linha um  
linha dois  
linha três
```

Explicação 4. O ponto-e-vírgula sempre significa fim de linha, não importa onde ele está. Isso significa que você pode escrever várias linhas de código em uma só. Além disso, se a linha for muito grande, você pode dar <ENTER> à vontade, visto que o compilador sabe que a linha só termina no ponto-e-vírgula.

Exemplo 1. Várias linhas de código em uma só. Abaixo temos cinco linhas, e não duas.

```
linha um; linha dois; linha três;  
linha quatro; linha cinco;
```

Exemplo 2. Agora, linhas de código que ocupam várias linhas. Abaixo, temos duas linhas de código, e não cinco.

```
esta é a linha um, e ela é muito  
grande para caber em uma linha;  
esta é a linha dois, que é maior  
ainda que a linha um, sendo preciso  
três linhas para completar o código;
```

Tópico 3. Blocos e escopo de variáveis

Explicação 5. Em C++, às vezes, temos estruturas, que são códigos que pertencem à uma linha de código. Esses blocos são delimitados por abre e fecha chaves. Um bloco pode ter qualquer quantidade de linha, mas é importante que seja delimitado por chaves. As linhas que iniciam os blocos não terminam com ponto-e-vírgula, por que isso significaria que o bloco chegou ao fim, o que não é verdade. Um bloco só termina quando fecha-se a chave.

Exemplo 1. Um exemplo simples: os cômodos estão dentro de casa. Por isso, casa inicia um bloco que contém os cômodos.

```

casa
{
quarto;
sala;
banheiro;
cozinha;
}

```

Exemplo 2. Não coloque ponto-e-vírgula na linha que inicia um bloco. Se você fizer isso, estará terminando o bloco antecipadamente.

```

casa;          //ERRO!
{
quarto;
sala;
banheiro;
cozinha;
}

```

O exemplo está errado. E também estaria errado se a gente colocasse o ponto-e-vírgula na última chave (}). Não devemos colocar ponto-e-vírgulas nesses casos.

Explicação 6. Variáveis são palavras que apontam para áreas da memória. Essas áreas armazenam informações. Por enquanto, assumo que uma variável é uma palavra que guarda algum valor. Em C++, existem vários tipos de variáveis: variáveis que guardam números, palavras etc. Nós dizemos que declaramos uma variável, quando criamos uma nova variável (que ainda não existe no escopo) para armazenar algum valor. Para que a gente possa declarar uma variável, é preciso primeiro escrever de que tipo ela é, e depois, o nome. O nome da variável nós escolhemos; podemos escolher o nome que a gente quiser. Isso não vai interferir na compilação do programa (compilação é o ato de o compilador transformar seu código em um programa executável).

Exemplo 1. No exemplo abaixo, estamos criando uma variável que vai armazenar um número inteiro. O tipo dessa variável é **int**, que vem da palavra “integer”, em inglês, que significa “inteiro”. Podemos escolher qualquer nome para a variável (dentro de algumas regras, é claro). No exemplo, vamos chamá-la de **x**, simplesmente. Então, como se faz? Primeiro escrevemos o tipo (que é **int**) e depois, o nome que a gente escolheu para a variável. E não devemos esquecer de terminar a linha com um ponto-e-vírgula¹.

```
int x;
```

Explicação 7. Podemos declarar (isto é, criar) várias variáveis no programa. E de diversos tipos. Além disso, podemos criar diversas variáveis do mesmo tipo, ao mesmo tempo.

Exemplo 1. Declaração de diversas variáveis, de tipos diferentes. A saber, **int** são variáveis que vão armazenar números inteiros, **float** armazenará valores decimais, ou seja, com ponto (e não vírgula, como é costume aqui no Brasil) e **char** vai armazenar caracteres, como por exemplo, 'a', 'c', '8', 'z'.

```
int variavel1;
float variavel2;
char variavel3;
```

Exemplo 2. Agora, a declaração de diversas variáveis do mesmo tipo. Basta usar vírgula entre um nome de variável e outra. Lembrando que o nome é você quem escolhe.

```
int x, y, z;
float a, b, c, d;
```

Explicação 8. Você não pode redeclarar variáveis. Isto é, não pode criar uma variável que já foi criada no mesmo escopo. O nome de uma variável já existente no escopo não pode ser usado para a criação de uma nova variável. O C++ **não** substitui uma variável existente.

Exemplo 1. Você não pode redeclarar variáveis que existem no mesmo escopo.

```
int x;
float y;
int x;           //ERRO! X existe
char y;         //ERRO! Y existe, e é do tipo float
```

¹ Está aí um belo exemplo de como precisamos de muitas linhas para explicar uma coisa pateticamente simples como declaração de variáveis. Coisas mais complexas às vezes exigem menos explicações do que coisas tão simples. Este exemplo sente-se lisonjeado.

Explicação 9. Um escopo de variável é o local onde a variável consegue ser vista. Um exemplo simples é de um aluno dentro de uma sala de aula. Suponhamos que hajam duas salas de aula, a sala A e a sala B. Na sala A existe um aluno chamado João. Isso significa que, se o professor chamar por João, este aluno atenderá. Se o professor que está na sala B chamar por João, não obterá resposta, pois o escopo de João não é na sala B, e sim, na sala A. Transportando isso para a programação C++, dizemos que uma variável é visível apenas em seu escopo. O escopo pode ser um bloco, por exemplo.

Exemplo 2. O exemplo das duas salas de aula.

```
sala A
{
  Pedro;
  João;
  Maria;
  Jonatas;
}

sala B
{
  Armanda;
  Brenda;
  Camila;
  Brutus;
}
```

Neste exemplo, os alunos que estão dentro da sala A não podem ver os alunos que estão dentro de sala B. E vice-versa. Então, o escopo de Pedro está na sala A, e o escopo de Brutus, na sala B.

Explicação 10. Foi dito anteriormente que uma variável não pode ser declarada onde existe outra com mesmo nome do mesmo escopo. Isso foi dito de propósito, pois fora do escopo a variável já não existe, e uma com o mesmo nome pode ser declarada sem problema.

Exemplo 1. Variáveis com o mesmo nome em escopos diferentes. Não há problema.

```
escopo A
{
  int var1, var2, var3;
}

escopo B
{
  int var3, var4;
}
```

Exemplo 2. Variáveis com o mesmo nome no mesmo escopo. Aí sim, há problema.

```
escopo A
{
  int var1, var2, var3;
  int var2; //ERRO. var2 existe.
}
```

Explicação 11. Variáveis de escopo inferior enxergam variáveis de escopo superior; mas variáveis de escopo superior não enxergam variáveis de escopo inferior. Dizemos escopo inferior quando, por exemplo, temos um bloco dentro de um bloco. As variáveis mais internas vêem as variáveis mais externas.

Exemplo 1. Abaixo, há um erro. O erro é indicado por um comentário //ERRO na mesma linha em que o erro acontece.

```
escopo A
{
  int a;
  int b;
  int c, d;

  escopo B
  {
    int d;    //ERRO! d existe
    int e, f;
    int g;
  }
}
```

Exemplo 2. Agora, não haverá nenhum erro, pois variáveis de um escopo mais externo, ou superior, não conseguem ver variáveis de escopo mais interno.

```
escopo A
{
  escopo B
  {
    int a, b;
  }

  int a;
}
```

Neste exemplo, embora a variável A existe no programa, quando nós declaramos int a pela segunda vez, o compilador já esquecer da existência da primeira variável a, pois o escopo já tinha terminado quando a chave anterior foi fechada.

Tópico 4. Tipos de dados e atribuição de valores

Explicação 12. Geralmente atribui-se valor a uma variável (que, é lógico, já tenha sido declarada; ela precisa existir para receber algum valor) colocando o nome da variável, o sinal de =, e o valor. O valor deve ser compatível com o tipo de variável. Não se pode atribuir à uma variável numérica um tipo caractere, e vice-versa. Você também pode atribuir um valor à variável no momento em que declara-a. Além disso, como o próprio nome sugere, uma “variável” pode ter seu valor alterado em qualquer momento do programa, desde que esta seja visível naquele escopo.

Exemplo 1. Formas corretas de se atribuir valores à variáveis.

```
int x;
x = 5;
char y;
y = 'z';
float a;
a = 5.5;
```

Neste exemplo, primeiro declaramos a variável. Uma vez que ela exista, podemos atribuir valores a elas. A variável **x** é do tipo **int**, portanto, atribuímos um número inteiro para ela. A mesma coisa a variável **y**, que recebeu um caractere, e a variável **a**, que recebeu um valor decimal.

Exemplo 2. Você pode atribuir valores às variáveis no momento da declaração das mesmas.

```
int x = 5;
int z = 8, f = 155;
float a = 3.14;
```

Exemplo 3. Uma forma errada de atribuir valores. Neste exemplo, a variável e o valor não são compatíveis.

```
int x = 'a'; //int vs caractere
char c = 55; //caractere versus int
int f = 3.14; //int versus float
```

Agora, atente para o seguinte fato: um número inteiro pode estar numa variável decimal, pois 3 é a mesma coisa que 3.0, por exemplo. Assim, isso está certo:

```
float t = 2;
```

Explicação 13. Uma variável pode receber o conteúdo de outra variável, desde que sejam compatíveis. Entretanto, o valor é copiado de uma para a outra, e as duas prosseguem com suas “vidas” daí para frente.

Exemplo 1. O modo correto de copiar valores de variáveis. No exemplo, a variável que está antes do igual “copia para si” o conteúdo da variável que está depois do sinal.

```
int x = 1;
int y = 3;
x = y; //x recebe 3

float a;
float b = 3.14;
a = b; //Ou seja, a recebe 3.14

b = 5.5; //b vale 5.5, mas a continua valendo 3.14
```

Tópico 5. Operações com variáveis numéricas

Explicação 14. Uma variável numérica (como **int** e **float**) pode receber o resultado de algum cálculo matemático envolvendo números ou outras variáveis, desde que sejam compatíveis, é claro. Os sinais são:

Sinal	Significado
+	Soma. $x = a + b;$
-	Subtração $x = a - b;$
*	Multiplicação $x = a * b;$
/	Divisão $x = a / b;$
()	Parênteses $x = (a + b) - (c * d);$
%	Resto da divisão. $x = a \% b;$

Exemplo 1. É perfeitamente inteligível o código abaixo.

```
int a = 5, b = 8, c = 2;
int x = ( a + b ) * ( b - ( 3 + c ) );
```

Se você precisar usar parênteses, tenha cuidado para não se esquecer de fechá-los todos, como neste exemplo.

Tópico 6. Tipos de erro que seus programas podem apresentar

Explicação 15. Erro de compilação. Ocorre quando a gente escreve o programa de maneira errada, esquecendo ponto-e-vírgulas, ou todas as outras coisas erradas dos exemplos citados. Quando dá um erro de compilação, o programa, obviamente, não é gerado pelo compilador, até você corrigir o erro.

Explicação 16. Falha de segmentação. Este erro o compilador não consegue detectar; portanto, o programa é gerado e executado, mas em algum momento ele termina bruscamente. É um erro bastante comum quando se está programando com vetores. Estudaremos vetores mais tarde neste curso.

Explicação 17. Outro tipo de erro. Acontece que o código digitado está correto, o programa é compilado e funciona completamente, só que o que acontece não é o que foi planejado. No caso, é mais um erro de planejamento mesmo, e o programador precisa ver em que ele errou. Não é erro de sintaxe: o programa funciona, mas não funciona como esperado.

Exemplo 1. O exemplo é bem simples. Imagine um programa que peça que o usuário digite um número, e depois, retorna ao usuário o dobro deste número. Entretanto, sem querer, o programador digitou, em vez de “dobro”, a palavra “triplo”. O programa vai fazer o cálculo corretamente, mas olha o que acontecerá:

```
Por favor, digite um número:    5  
O triplo de 5 é 10.
```

No caso, o programa realmente calcula o dobro do número, mas a mensagem que é exibida contém a palavra “triplo”.