

Utilizando ponteiros em C e C++

prof. Marcio Feitosa

O que veremos a seguir é uma breve introdução aos ponteiros em C e C++.

Todo algoritmo trabalha com dados. Esses dados, na maioria dos casos, é fornecido ao algoritmo que, por sua vez, irá fazer algum tipo de trabalho com eles.

Dentro desse algoritmo, os dados deverão ser acondicionados em variáveis para que possam ser trabalhados. Essas variáveis podem conter diversos tipos de dados, sendo os mais comuns os números, as cadeias de caracteres e as datas.

As linguagens C e C++ são ditas "tipadas", ou seja, ao declarar uma variável é necessário especificar o tipo do dado (*data type*) que será armazenado. Por exemplo:

- `int x;`
- `float y;`
- `char x;`
- `string s;` (*somente C++*)

A variável declarada como `int` (numérico inteiro) só poderá armazenar números inteiros. No entanto, o conteúdo da variável em si pode conter duas categorias (as duas do mesmo *data type*) de valores:

- o valor do dado.
- o endereço de onde está a variável que contém o dado.

Todo dado só é diretamente acessível dentro do escopo do método, ou seja, entre as chaves delimitadoras `{}`. Ou, em alguns casos, se a variável for declarada como global.

Em diversas situações é necessário acessar e também modificar o estado de variáveis que estejam em outro escopo. Para isso é preciso criar um ponteiro.

Analisemos o seguinte código:

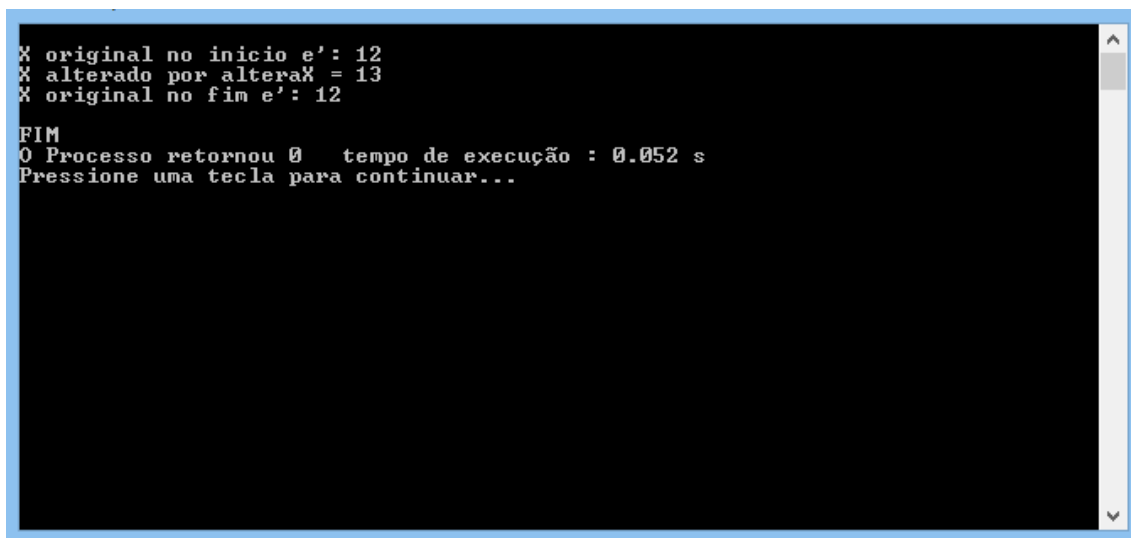
Utilizando ponteiros em C e C++

prof. Marcio Feitosa

```
1  #include <iostream>
2
3  using namespace std;
4
5  int alteraX(int k) {
6      k += 1;
7      return k;
8  }
9
10 int main() {
11
12     int x = 12;
13     cout << "\nX original no inicio e': " << x;
14     int xAlterado = alteraX(x);
15     cout << "\nX alterado por alteraX = " << xAlterado;
16     cout << "\nX original no fim e': " << x;
17     cout << "\n\nFIM";
18
19     return 0;
20 }
21
```

Figura 1

Se executarmos este código, receberemos a seguinte tela:



```
X original no inicio e': 12
X alterado por alteraX = 13
X original no fim e': 12

FIM
0 Processo retornou 0 tempo de execução : 0.052 s
Pressione uma tecla para continuar...
```

Figura 2

O método main declara a variável x na linha 12 e atribui o valor 12 (coincidência), imprime o valor, depois chama o método alteraX (acima do main) enviando x que faz apenas o incremento de uma unidade no valor do parâmetro enviado. Imprime o valor retornado por alteraX e finalmente imprime (novamente) o valor de x.

Utilizando ponteiros em C e C++

prof. Marcio Feitosa

Observamos que o valor original de `x` não foi alterado. Isto porque o método chamador, no caso o método `main`, faz a chamada por valor, e não por referência (usando ponteiros). O que ocorreu é que o método `alteraX` recebeu uma cópia de `x` (que nele passa a ser conhecida por `k`) e não a `x` propriamente dita. Então, toda e qualquer alteração neste valor recebido não interfere em nada no valor de `x`.

Mas, e se desejássemos um método que fizesse as alterações em `x`. Obviamente que poderíamos fazer:

```
x = alteraX(x);
```

Resolveria o problema da forma mais simples possível. Mas existem situações em que fica burocrático ou com mais uso de memória com a atividade de enviar e retornar a cópia (alterada) de volta. Também há casos em que não é possível enviar o parâmetro por valor, como é o caso dos arrays e outros tipos de objetos, somente por referência.

Se fizermos as seguintes alterações:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int alteraX(int * k) {
6      * k += 1;
7      return * k;
8  }
9
10 int main() {
11
12     int x = 12;
13     int * px = & x;
14     cout << "\nX original no inicio e': " << x;
15     int xAlterado = alteraX(px);
16     cout << "\nX alterado por alteraX = " << xAlterado;
17     cout << "\nX original no fim e': " << x;
18     cout << "\n\nFIM";
19
20     return 0;
21 }
```

Figura 3

Na linha 13 do método `main` foi criada uma variável ponteiro (`px`). À frente desta variável vai o sinal `*` (asterisco) que

Utilizando ponteiros em C e C++

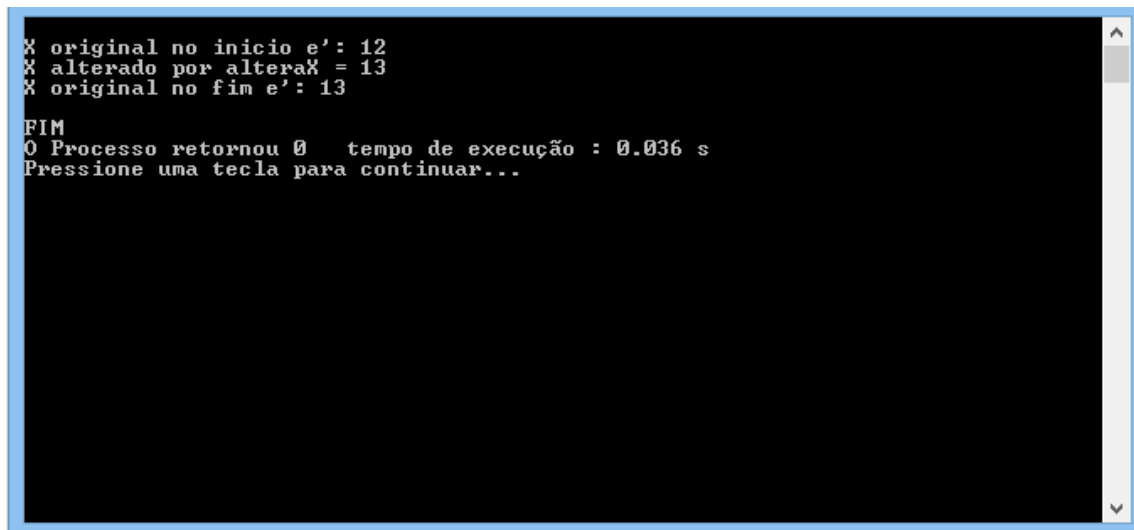
prof. Marcio Feitosa

indica que se trata de um ponteiro, ou seja, vai acondicionar endereços.

Atenção para não confundir com operações de multiplicação que utilizam o mesmo sinal (*).

Ao mesmo tempo que declaramos a variável ponteiro px, atribuímos a ela o endereço da variável x. O que é feito pelo operador &.

O método alteraX foi alterado para operar com uma variável ponteiro. A tela de saída é a seguinte:

A screenshot of a terminal window with a black background and white text. The text shows the execution of a program that uses pointers. It prints the initial value of a variable, the value after being modified by a function, and the final value. It also shows the end of the program and the execution time.

```
% original no inicio e': 12
% alterado por alteraX = 13
% original no fim e': 13

FIM
O Processo retornou 0 tempo de execução : 0.036 s
Pressione uma tecla para continuar...
```

Figura 4

Observamos que o valor inicial de x foi alterado, ou seja, o método alteraX acessou a variável x que pertence ao escopo do método main.

Existe uma forma mais simples de se fazer isso. Observemos o código abaixo:

Utilizando ponteiros em C e C++

prof. Marcio Feitosa

```
1  #include <iostream>
2
3  using namespace std;
4
5  int alteraX(int &k) {
6      k += 1;
7      return k;
8  }
9
10 int main() {
11
12     int x = 12;
13     cout << "\nX original no inicio e': " << x;
14     int xAlterado = alteraX(x);
15     cout << "\nX alterado por alteraX = " << xAlterado;
16     cout << "\nX original no fim e': " << x;
17     cout << "\n\nFIM";
18
19     return 0;
20 }
21
```

Figura 5

A única modificação em relação ao primeiro código (Figura 1) é a inclusão do operador & no parâmetro do método alteraX. A resultante é a mesma da Figura 4.

A diferença neste último caso é que o método alteraX não terá o poder de remover a variável x (nele chamada de k) para liberar memória. E muitas vezes isso é conveniente.

-0-0-0-0-0-0-0-