

Minicurso de C++

Aula 03

Orientação a Objetos: Classes - Parte 1

Responsáveis:

Gabriel Daltro Duarte

Rafael Guerra de Pontes

Walisson da Silva

Wesley da Cunha Santos

Ianes Grécia



Programação Orientada a Objetos

Objetos

- Onde quer que você olhe no mundo real, você vê objetos - pessoas, animais, plantas, carros, aviões, edifícios, computadores e assim por diante.
- Os homens pensam em termos de objetos. Telefones, casas, sinais de trânsito, fornos de microondas, são apenas mais alguns objetos que vemos durante o dia.

Objetos

- Os objetos se dividem em duas categorias: Os animados e os inanimados.
- Os animados são, são em certo sentido, objetos vivos - eles se movem por conta própria. Os inanimados não se movem por conta própria.
- Em ambos os casos os objetos possuem **atributos** (tamanho, forma, cor, peso, etc) e exibem **comportamentos** (por exemplo, uma bola bate, infla, seca; o bebê chora, ri, anda, engatinha).

Objetos

- Os humanos aprendem sobre os objetos existentes estudando seus *atributos* e observando seu *comportamento*.
- Os programas são construídos modelando os objetos através de seus atributos, comportamentos e inter-relacionamentos, assim como descrevemos objetos do mundo real.

Objetos

- Os dados e as operações são encapsulados (empacotados) em objetos. Os atributos (dados) e as operações são intimamente ligados contextualmente.
- Assim como as pessoas trocam mensagens entre si, os objetos também se comunicam via mensagens. Por exemplo, um objeto conta bancária pode receber a mensagem pra reduzir seu saldo em certa quantia porque o cliente retirou essa quantia em dinheiro.

Classes e Objetos

- Os objetos são definidos através das classes. As classes são tipos de dados definidos pelo usuário.
- Dentro das classes há variáveis que representam os **atributos** do objeto e há funções que representam as **operações** (comportamentos) que o objeto é capaz de realizar.

Classes e Objetos

- Por exemplo, uma classe conta bancária pode possuir os atributos número da conta e saldo da conta e pode possuir a ação diminuir saldo.
- Os atributos número da conta e saldo da conta são representados por variáveis e a operação *diminuir saldo* é representada por uma função que altera o conteúdo da variável *saldo da conta*.
- Em C++ as variáveis contidas dentro de uma classe são chamadas **membros de dados** e as funções de uma classe são chamadas **funções-membro**.

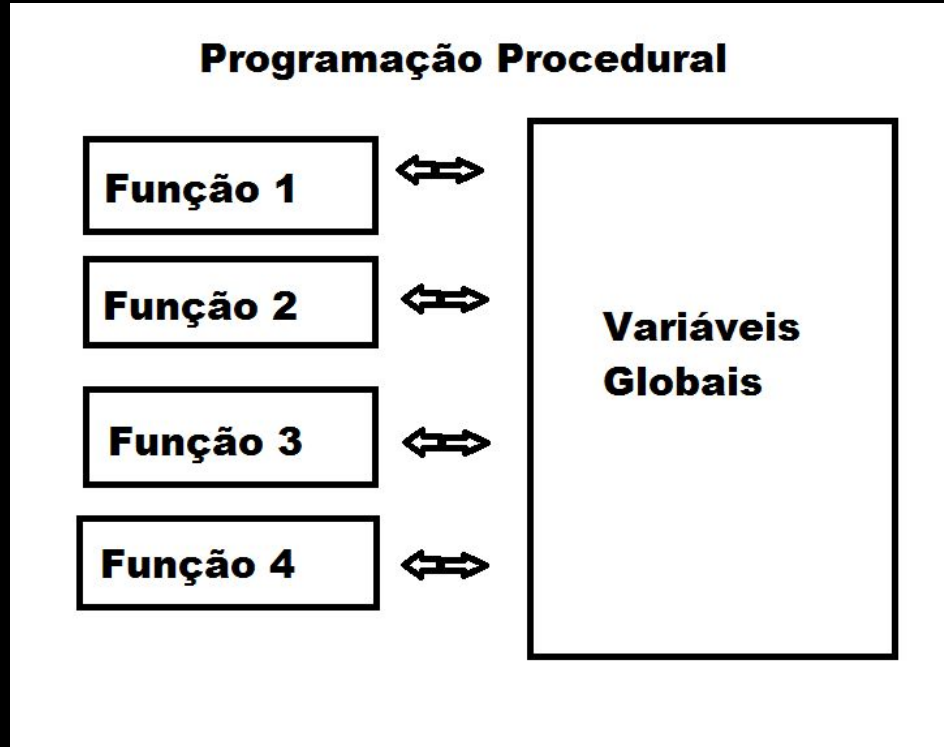
Classes e Objetos

- As classes estão para os objetos assim como as plantas arquitetônicas estão para as casas.
- Uma classe é um plano para criar um objeto de uma classe.
- Da mesma maneira que podemos construir várias casas a partir da mesma planta, podemos *instanciar* (criar) muitos objetos a partir de uma mesma classe.
- Você não pode dormir no quarto de uma planta, você só pode dormir no quarto de uma casa real!!!

POO x Programação procedural

- Na programação procedural (como, por exemplo, C) a programação tende a ser orientada a ação.
- A unidade de programação é a função;
- Programadores concentram-se em escrever funções;
- Os programadores agrupam as ações que realizam alguma tarefa comum em funções e agrupam essas funções formando o programa.
- Os dados são certamente importantes em C, mas é perceptível que eles existem principalmente em suporte às ações realizadas pelas funções.

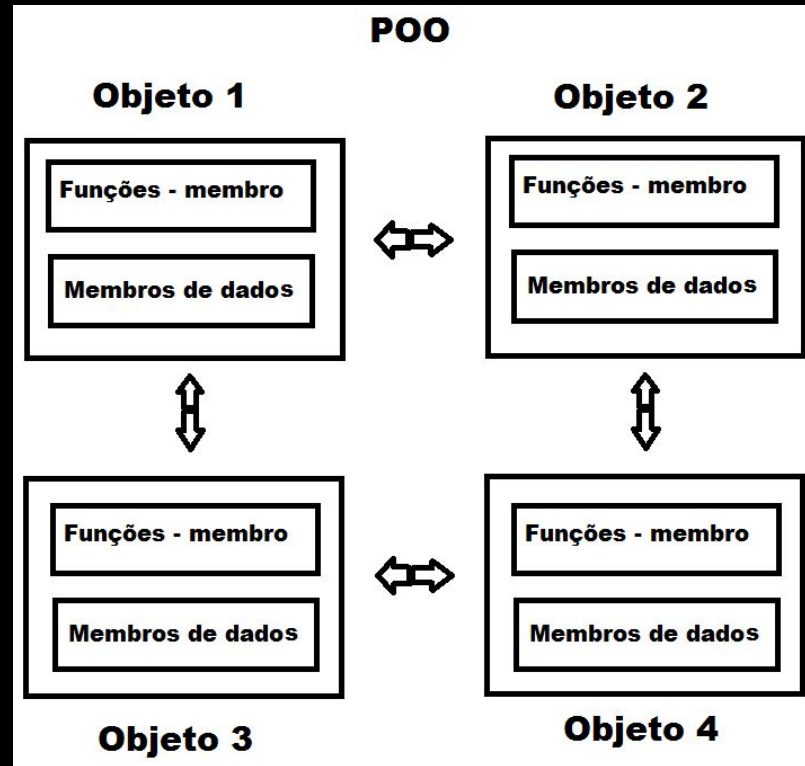
POO x Programação procedural



POO x Programação procedural

- Programadores C++ concentram-se em criar classes.
- Os programadores utilizam tipos predefinidos e outros tipos definidos pelo usuário como os blocos de construção de novos tipos de dados definidos pelo usuário (classes);

POO x Programação procedural



Declaração de uma Classe

```
class NomeDaClasse {  
    public:  
        //(atributos e funções membros públicas)  
    private:  
        //(atributos e funções membros privados)  
    protected:  
        //(atributos e funções membros protegidos)  
};
```

Especificadores de Acesso

`public` ⇒ funções e variáveis membros podem ser acessadas em qualquer parte do programa. Variáveis podem, inclusive, ser modificadas.

`private` ⇒ funções e variáveis membros não podem ser sequer vistas fora da classe.

`protected` ⇒ análoga ao modificador de acesso privado, mas funções e variáveis membros podem ser acessados por classes filhas (ou classes derivadas)

Exemplo

```
#include <iostream>
#include <string>
using namespace std;
class Gato {
public:
    int idade;
    string raca;
    string nome;
    float peso;
    void exibirCaracteristicas(){
        cout << "Tenho um gato " << raca;
        cout << " chamado " << nome;
        cout << ", com " << idade;
        cout << " anos de idade e ";
        cout << peso << " kg.\n";
    }
};
```

```
int main(){
    Gato meuGato;
    meuGato.idade = 3;
    meuGato.raca = "Persa";
    meuGato.nome = "Bob";
    meuGato.peso = 5.2;
    meuGato.exibirCaracteristicas();
    return 0;
}
```

Saída

C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\exClasse01.exe

Tenho um gato Persa chamado Bob, com 3 anos de idade e 5.2 kg.

Classe Gato

- Observe que as variáveis idade, raca, nome, peso e a função `exibirCaracteristicas ()` são public. Assim elas podem ser acessíveis em qualquer parte do programa em que um objeto da classe Gato tenha sido instanciado.
- O objeto `meuGato` foi instanciado na função `main`, assim todos os membros de dados e funções public podem ser acessíveis na função `main`.

Classe Pessoa

```
#include <iostream>
```

```
#include <string> // classe string padrão do C++
```

```
// Declaração da classe Pessoa
```

```
class Pessoa
```

```
{
```

```
public: // especificador de acesso
```

```
    void setNome (std::string n )
```

```
    {
```

```
        nome = n;
```

```
    }
```

```
    std::string getNome(void)
```

```
    {
```

```
        return nome; // retorna o conteúdo do membro de dado private nome
```

```
    }
```

```
    void displayNome(void) //Função que exibe o nome da pessoa
```

```
    {
```

```
        std::cout<<"Nome: " <<getNome() << std::endl;
```

```
    }
```

```
private:
```

```
    std::string nome;
```

```
};
```

```
int main ()
```

```
{
```

```
    Pessoa Eu;
```

```
    Eu.setNome("GABRIEL");
```

```
    Eu.displayNome();
```

```
    return (0);
```

```
}
```

Classes

- A declaração da maioria dos membros de dados é feita após o especificador de acesso `private`.
- Variáveis e funções declaradas após o especificador de acesso `private` só são acessíveis a funções-membro da classe na qual elas estão declaradas.
- Na classe `pessoa`, por exemplo, `nome` foi declarado como `private`, assim só pode ser acessado pelas funções membro `setNome ()`, `getNome ()` e `displayNome ()`. Ele não pode ser acessado por exemplo pela `main`.

Classes

- As classes costumam fornecer funções membro public para permitir a clientes da classe configurar (set ou seja, atribuir valores) ou obter (get, ou seja, obter valores de) membros de dados private.
- Funções set e get de uma classe também devem ser usadas por outras funções membro dentro da classe para manipular os dados private da classe, embora essas funções possam acessar os dados private diretamente. Como por exemplo a função displayNome(). Dessa forma criamos classes mais robusta(mais fácil de manter e fazer manutenções).

Classes

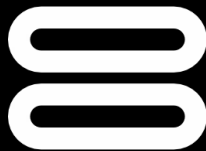
- Como regra geral, funções membro são declaradas como public e membros de dados são declarados como private
- Se nenhum especificador de acesso por definido, os membros de uma classe são private por padrão, mas é sempre bom explicitar o tipo de acesso que os membros da classe possuem.

Classes

- Do ponto de vista como os objetos ficam armazenados na memória, cada objeto possui sua própria cópia dos membros de dados, mas as instruções das funções membro só possuem uma cópia que é compartilhada com todos os objetos.

Exemplo de implementações externas ao bloco da classe

```
class Classe {  
    public:  
        void salutar();  
};  
  
Classe::salutar(){  
    cout << "Oi!\n";  
}
```



```
class Classe {  
    public:  
        void salutar(){  
            cout << "Oi!\n";  
        }  
};
```

Encapsulamento

- O encapsulamento consiste em impedir o acesso indevido a alguns atributos e/ou métodos de uma classe.
- Em C++, os níveis de encapsulamento – direitos de acesso – são indicados através de três palavras-chaves: *public*, *private* e *protected*.
- Por *default* todos os membros (dados e métodos) de uma classe são *private*.

Get e Set - Métodos Acessores

- O encapsulamento "protege" os atributos ou métodos dentro de uma classe, portanto devemos prover meios para acessar tais membros quando eles são privados, ou seja, quando possuem o modificador *private*.
- O que torna isso possível é a criação de métodos.

Get e Set - Métodos Acessores

- Em programação orientada a objetos, esses métodos são chamados de métodos acessores ou getters e setters, pois eles provêm acesso aos atributos da classe, e geralmente, se iniciam com get ou set, daí a origem de seu nome.

Set

- Usado para modificar algum campo ou atributo de uma classe;
- Se não for criado *set* para algum atributo, isso quer dizer que este atributo não deve ser modificado.
- Não é necessário que este método retorne nenhum valor, por isso, os métodos *setters* são void. Porém, obrigatoriamente, recebem um argumento que será o novo valor do atributo.

Get

- Usado para verificar algum campo ou atributo de uma classe.
- Como este método irá verificar um valor, ele sempre terá um retorno como *string*, *int*, *float*, etc. Mas não terá nenhum argumento.

```
#include <iostream>
```

```
using namespace std;
```

```
class TV {
```

```
private:
```

```
    int tamanho;
```

```
    int canal;
```

```
    int volume;
```

```
public:
```

```
    TV(int t, int c, int v){
```

```
        tamanho = t;
```

```
        canal = c;
```

```
        volume = v;
```

```
    }
```

```
    int getTamanho(){
```

```
        return tamanho;
```

```
    }
```

```
void setTamanho(int t){
```

```
    this->tamanho = t;
```

```
}
```

```
// ...
```

```
};
```

```
int main(){
```

```
    TV tv(40, 12, 22);
```

```
    tv.setTamanho(42);
```

```
    cout << tv.getTamanho() << endl;
```

```
    return 0;
```

```
}
```



Qual é a saída desse programa???

Exercício

Crie uma classe chamada Caixa, que possua os atributos altura, largura e profundidade. Além disso, que tenha uma função membro que retorna o volume dela.

Declare os atributos private e defina funções set e get public para alterar o conteúdo dos membro de dados.

Construtor

É uma função membro que:

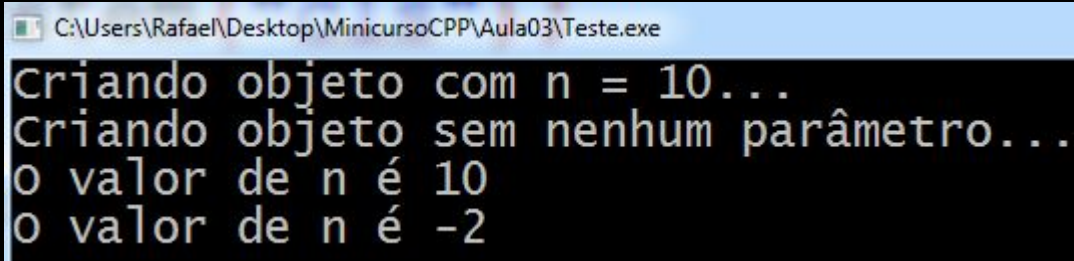
- Possui o mesmo nome da classe.
- Não possui tipo de retorno (sequer `void`).
- Usado para inicializar objetos com valores para alguns de seus membros.
- Por padrão, não possui nenhum parâmetro.
- Pode ser sobrecarregado.

```

#include <iostream>
#include <cstdlib>
using namespace std;
class Teste {
private:
    int n;
public:
    Teste (void){
        cout << "Criando objeto sem nenhum
parâmetro...\n";
    }
    Teste (int x){
        this->n = x;
        cout << "Criando objeto com n = ";
        cout << this->n << "... \n";
    }
    void imprimeN(){
        cout << "O valor de n é " << n << endl;
    }
};

```

Saída



```

C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\Teste.exe
Criando objeto com n = 10...
Criando objeto sem nenhum parâmetro...
O valor de n é 10
O valor de n é -2

```

```

int main(){
    system("chcp 1252");
    system("cls");

    Teste teste(10);
    Teste teste2;

    teste.imprimeN();
    teste2.imprimeN();

    return 0;
}

```

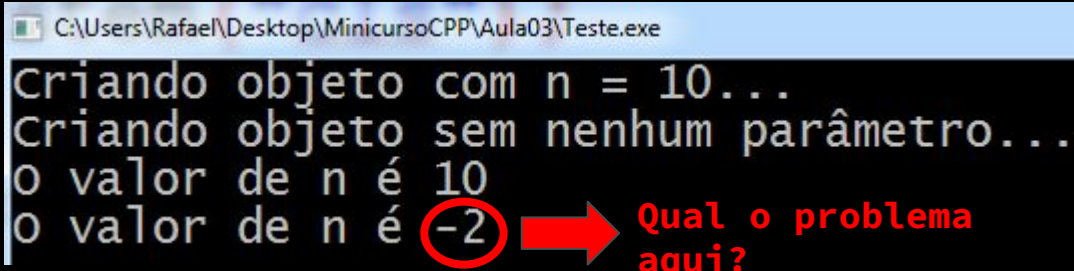


```

#include <iostream>
#include <cstdlib>
using namespace std;
class Teste {
private:
    int n;
public:
    Teste (void){
        cout << "Criando objeto sem nenhum
parâmetro...\n";
    }
    Teste (int x){
        this->n = x;
        cout << "Criando objeto com n = ";
        cout << this->n << "... \n";
    }
    void imprimeN(){
        cout << "O valor de n é " << n << endl;
    }
};

```

Saída



```

C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\Teste.exe
Criando objeto com n = 10...
Criando objeto sem nenhum parâmetro...
O valor de n é 10
O valor de n é -2

```

Qual o problema aqui?

```

int main(){
    system("chcp 1252");
    system("cls");

    Teste teste(10);
    Teste teste2;

    teste.imprimeN();
    teste2.imprimeN();

    return 0;
}

```

Sintaxe alternativa de construtor

```
#include <iostream>
using namespace std;
class Produto{
private:
    int id;
public:
    Produto(int id);
};
Produto::Produto(int i){
    this->id = i;
    cout << "Construindo objeto com id ";
    cout << id << ".\n";
}
int main(){
    Produto p(123456);
    return 0;
}
```



```
#include <iostream>
using namespace std;
class Produto{
private:
    int id;
public:
    Produto(int id);
};
Produto::Produto(int i): id(i)
{
    cout << "Construindo objeto com id ";
    cout << id << ".\n";
}
int main(){
    Produto p(123456);
    return 0;
}
```



C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\construtorAlternativo.exe

construindo objeto com id 123456.

C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\construtorAlternativo2.exe

construindo objeto com id 123456.

```
#include <iostream>
```

```
using namespace std;
```

```
class Produto{
```

```
private:
```

```
    int a, b;
```

```
    double c;
```

```
public:
```

```
    Produto(int, int, double);
```

```
};
```

```
Produto::Produto(int aa, int bb, double cc): a(aa), b(bb),
```

```
c(cc)
```

```
{
```

```
    cout << "Construindo objeto com a = " << a;
```

```
    cout << ", b = " << b << " e c = " << c << ".\n";
```

```
}
```

```
int main(){
```

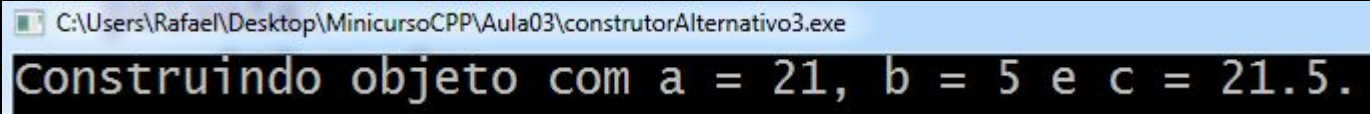
```
    Produto p(21, 5, 21.5);
```

```
    return 0;
```

```
}
```

Sintaxe alternativa de construtor para vários atributos

Saída



```
C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\construtorAlternativo3.exe
```

```
Construindo objeto com a = 21, b = 5 e c = 21.5.
```

Destruitor

- Função-membro especial que é executada sempre que um objeto da classe em questão sai de escopo ou quando a expressão de deleção é aplicada ao ponteiro desse objeto.
- Possui o mesmo nome da classe, mas com o prefixo '~'.
- Não possui parâmetros.
- Não pode retornar nenhum tipo de dado.

```
// Exemplo de classe com Destrutor
```

```
#include <iostream>
```

```
using namespace std;
```

```
class ClasseDestruidora{
```

```
private:
```

```
    int a, b;
```

```
public:
```

```
    ClasseDestruidora(int, int);
```

```
    ~ClasseDestruidora();
```

```
};
```

```
ClasseDestruidora::ClasseDestruidora(int aa, int bb): a(aa), b(bb)
```

```
{
```

```
    cout << "Construindo objeto com a = " << a;
```

```
    cout << " e b = " << b << "...\n";
```

```
}
```

```
ClasseDestruidora::~~ClasseDestruidora(){
```

```
    cout << "Destruindo o objeto!\n";
```

```
}
```

```
int main(){
```

```
    ClasseDestruidora objeto(21, 52);
```

```
    return 0;
```

```
}
```



C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\destrutor.exe

Construindo objeto com a = 21 e b = 52.
Destruindo o objeto!

Saída

Exercício

Crie uma classe que representa um ponto no plano cartesiano. Crie uma função membro que retorne a distância desse ponto à origem do plano. Leia os pontos do usuário. invoque a função para exibir essa distância à origem.

Dica: importe a biblioteca matemática `<cmath>` e use a função `sqrt(x)`.

Friend Functions

- Funções “amigas” da classe, mas que não pertencem a ela.
- Estão fora do escopo da classe.
- Podem acessar todos os atributos e funções membros da classe, mesmo que `private` ou `protected`.
- Basta colocar a palavra `friend` antes do protótipo da função, na declaração da classe.

```
//Exemplo com friend function
```

```
#include <iostream>
```

```
using namespace std;
```

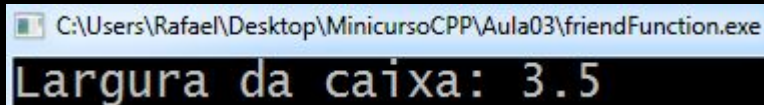
```
class Box {  
    double width;  
public:  
    friend void printWidth( Box box );  
    void setWidth( double wid );  
};
```

```
void Box::setWidth( double wid ) {  
    width = wid;  
}
```

```
void printWidth( Box box ){  
    cout << "Largura da caixa: " << box.width << endl;  
}
```

```
int main( ){  
    Box box;  
    box.setWidth(3.5);  
    printWidth( box );  
    return 0;  
}
```

Saída



C:\Users\Rafael\Desktop\MinicursoCPP\Aula03\friendFunction.exe
Largura da caixa: 3.5

Exemplo de classe: `string`

- Para usá-la, precisa-se incluir a biblioteca `<string>`.
- Aloca memória dinamicamente, de acordo com a necessidade.
- Possui diversos métodos para manipular e acessar dados sobre a string armazenada.
- Possui 7 construtores sobrecarregados (vide site <http://www.cplusplus.com/reference/string/string/string/>).

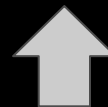
```
// Exemplo de programa que  
// demonstra diversos  
// construtores da classe string
```

```
#include <iostream>  
#include <string>
```

```
using namespace std;
```

```
int main(){  
    string s0("string inicial.");  
    string s1(s0);  
    string s2(s0,7,3);  
    string s3;  
    string s4(10,'A');  
    string s5(s0.begin(),s0.begin()+6);  
  
    cout << "s0: \"" << s0 << "\".\n";  
    cout << "s1: \"" << s1 << "\".\n";  
    cout << "s2: \"" << s2 << "\".\n";  
    cout << "s3: \"" << s3 << "\".\n";  
    cout << "s4: \"" << s4 << "\".\n";  
    cout << "s5: \"" << s5 << "\".\n";  
  
    return 0;  
}
```

```
rafael@rafael-inspiron-7520:~/Desktop/Rafael/MinicursoCPP/Aula03$ ./ExemploString  
s0: "string inicial."  
s1: "string inicial."  
s2: "ini".  
s3: "".  
s4: "AAAAAAAAAA".  
s5: "string".  
rafael@rafael-inspiron-7520:~/Desktop/Rafael/MinicursoCPP/Aula03$
```



Saída

```

#include <iostream>
#include <string>

using namespace std;

int main(){
    string s0("123456");
    string s1("123.456");
    string s2;
    if(s2.empty()){
        cout << "A s2 está vazia no início da
main.\n";
    }
    s2 = "Atribuindo dados com o operador =";
    if(s2.empty() == false){
        cout << "A s2 não está mais vazia após
a atribuição.\n";
    }
    string s3("abcdefghijklmnopqrstuvwxyz");
    string s4("Pedra");
    string s5("aaaaabbbbccccddddddeeeee");
    cout << "s0: \"\" << s0 << "\".\n";
    cout << "s1: \"\" << s1 << "\".\n";
    cout << "s2: \"\" << s2 << "\".\n";
    cout << "s3: \"\" << s3 << "\".\n";
    cout << "s4: \"\" << s4 << "\".\n";
    cout << "s5: \"\" << s5 << "\".\n";

```

```

        cout << "Chamada de algumas funções da
classe:\n";

        cout << "s3.at(0): \"\" << s3.at(0) << "\".\n";
        cout << "s3.at(s3.length()-1): \"\" <<
s3.at(s3.length()-1) << "\".\n";
        s4 += " molhada";
        cout << "Após s4 += \" molhada\", s4: \"\" << s4
<< "\".\n";

        s3.erase(4,10);
        cout << "Após s3.erase(4,10), s3: \"\" << s3 <<
\"\".\n";
        int pos = s5.find("ddddd");
        if(pos != string::npos){
            cout << "Encontrei a substring \"ddddd\" na
s5 na posição ";
            cout << pos << endl;
            cout << "Substituído dddd por
PRATO...\n";
            s5.replace(pos,5,"PRATO");
            cout << "s5: \"\" << s5 << "\".\n";
        }

        return 0;
    }
}

```

Saída

```
rafael@rafael-inspiron-7520:~/Desktop/Rafael/MinicursoCPP/Aula03$ ./ExemploString2
A s2 está vazia no início da main.
A s2 não está mais vazia após a atribuição.
s0: "123456".
s1: "123.456".
s2: "Atribuindo dados com o operador =".
s3: "abcdefghijklmnopqrstuvwxyz".
s4: "Pedra".
s5: "aaaaabbbbccccddddddeeeee".
Chamada de algumas funções da classe:
s3.at(0): 'a'.
s3.at(s3.length()-1): 'z'.
Após s4 += " molhada", s4: "Pedra molhada".
Após s3.erase(4,10), s3: "abcdopqrstuvwxyz".
Encontrei a substring "dddd" na s5 na posição 15
Substituído ddddd por PRATO...s5: "aaaaabbbbccccPRATOeeeeee".
rafael@rafael-inspiron-7520:~/Desktop/Rafael/MinicursoCPP/Aula03$
```

Exercício

Faça um programa que leia duas strings. Procure quantas vezes a segunda ocorre na primeira e substitua todas as ocorrências dessa segunda strings por 'X'.

Por hoje é só!
Bons estudos! :)