C++

Algoritmos e Estruturas de Dados I

Prof. Lucas Astore

O surgimento de C++

- Criada no Bell Labs em 1983
- Por Bjarne Stroustrup
- Possui a performance de C
- E as funcionalidades de outras linguagens como Simula e Algol
- Padronizada apenas em 1997

Bjarne Stroustrup

Comparativo entre c e c++

C	C++
Estruturada	Orientada a Objetos
malloc e **calloc**	**new**
free	**delete**
Passagem por valor	Passagem por valor ou referência
stdio	**iostream**
Variáveis declaradas no início de um bloco	Variáveis declaradas em **qualquer** parte do bloco

Comparativo entre c e c++

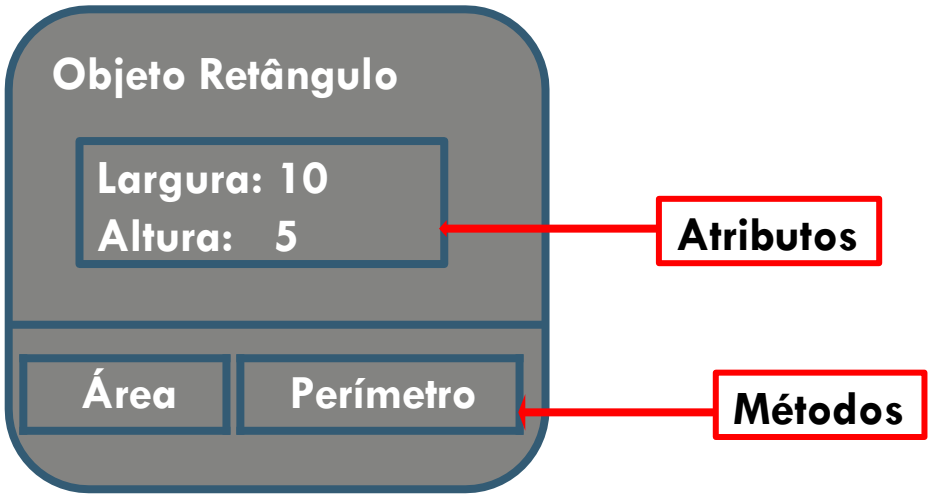
C	C++
Inteiro como valor booleano	Tipo **bool**
Duas funções não podem ter o mesmo nome	Duas funções não podem ter o mesmo protótipo
Argumentos são sempre necessários	Valor defaut para os argumentos
Casts simples	Novos tipos de cast
string como array de caracteres	Tipo string

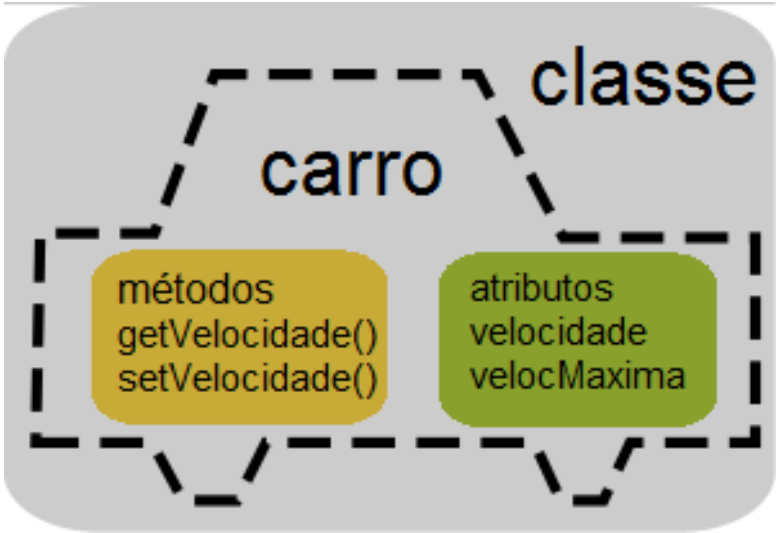
Estrutura básica

```
#include <iostream>

using namespace std;

//definição de constantes
//funções

int main()
{
//declaração de variáveis

//sentenças

}
```

Orientação a Objetos

Analogia:





Orientação a Objetos

- Objetos são tipos definidos pelo usuário
- Eles podem ter:
  - **Atributos** - são as informações que um objeto guarda
  - **Métodos** - são as funções que determinam o seu comportamento

Classes

- São definições a partir das quais os objetos podem ser criados
- As classes determinam quais são os atributos e métodos de um objeto

Sintaxe:

```
class nomeDaClasse {
   corpoDaClasse;
};
```

Um retângulo em C e C++

Em C:

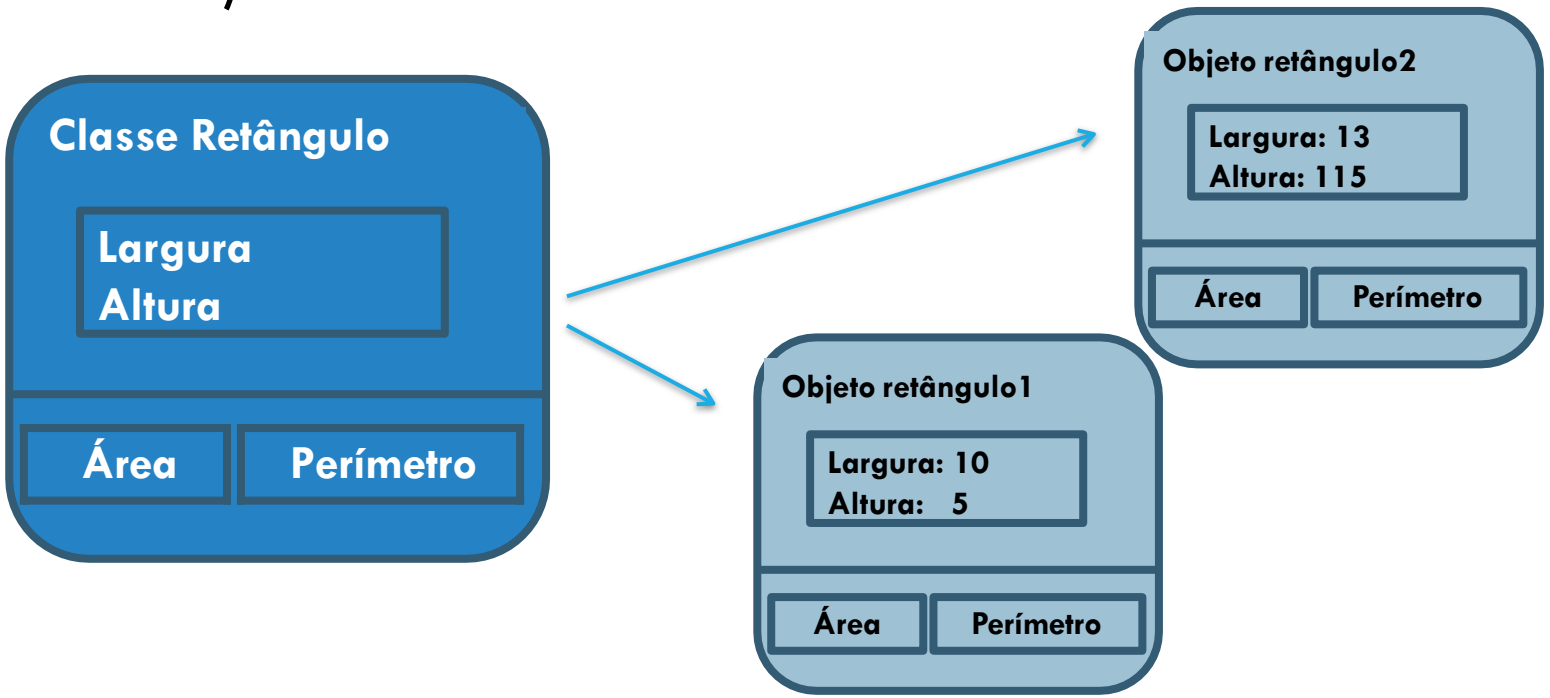
```c
struct Retangulo {
    int altura;
    int largura;
};

int area(struct Retangulo *r)
{
    return r.altura * r.largura;
}
int perimetro(struct Retangulo *r) {

    return 2 * (r.largura + r.altura);
}
```

Em C++:

```c
class Retangulo {
    int largura;
    int altura;

    int area() {
        return largura * altura;
    }

    int perimetro() {
        return 2 *(largura + altura);
    }

}
```

Diferença entre Classe e Objeto

- Classe é apenas a descrição de um tipo de objeto (“receita do bolo”)
- Objetos são as instâncias de uma classe (“bolos feitos com a receita”)



Categorias de Permissão

- Membros de uma classe podem ser:

  public

  - Podem ser acessados em qualquer lugar

  private

  - Só podem ser acessados pelos membros da própria classe

  protected

  - Podem ser acessados apenas por membros da própria classe ou das suas sub-classes

Exemplo

```
class Retangulo {
    int largura;

    private:
        int altura;

    public:
        int area() {
            return largura * altura;
        }

    protected:
        int perimetro() {
            return 2 *(largura + altura);
        }
};
```

Obs.: Por default todo membro de uma classe é considerado `private`

Exemplo

```
class Retangulo {
    int largura;

    private:
        int altura;

    public:
        int area() {
            return largura * altura;
        }

    protected:
        int perimetro() {
            return 2 *(largura + altura);
        }
};
```

Obs.: Por default todo membro de uma classe é considerado `private`

```
int main() {
    Retangulo r;

    // Errado:
    r.altura = 10;
    // Errado:
    r.largura = 40;

    // OK:
    int a = r.area();

    // Errado:
    a = r.perimetro();
}
```

O operador ::

- Permite a implementação de métodos fora da classe
- A classe passa a possuir apenas o protótipo do método
- O corpo pode ficar no mesmo arquivo ou em outro
- Sintaxe:

```
nomeDaClasse::nomeDoMembro
```

Exemplo

```
class Retangulo {
    private:
    int largura;
    int altura;

    public:
    int area();
    int perimetro();

};

int Retangulo::area() {
    return largura * altura;
}
int Retangulo::perimetro() {
    return 2 *(largura + altura);
}
```

Construtor

- É um método especial que é chamado quando criamos um novo objeto
- Deve possuir o mesmo nome da classe
- Não possui retorno
- É utilizado para inicializar os atributos da classe

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        //Construtor:
        Retangulo(int a, int l){
            altura = a;
            largura = l;
        }
};
```

ou assim:

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        Retangulo(int a, int l);
};

Retangulo::Retangulo(int a, int l) {
    altura = a;
    largura = l;
}
```

Destrutor

- Método especial que é chamado automaticamente quando um objeto está prestes a ser apagado da memória
- Deve ter o mesmo nome da classe mas precedido por um ~
- Assim como o construtor ele não possui retorno
- Além disso, ele não pode ter parâmetros

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        Retangulo(int a, int l);
        ~Retangulo() { } // destrutor padrão
};

Retangulo::Retangulo(int a, int l) {
    altura = a;
    largura = l;
}
```

Métodos Get e Set

get()

Serve para se ter acesso aos atributos encapsulados de uma classe

Exemplo:

```
int getLargura() { return largura; }
```

set()

Útil para permitir a modificação dos atributos da classe encapsulados

Exemplo:

```
void setLargura(int l) {
    largura = l;
}
```

Exemplo

```
class Retangulo {
    private:
        int largura;
        int altura;
    public:
        int getAltura() { return altura;}
    protected:
        void setAltura(int a) {
            // evita um valor inválido
            if (a > 0) altura = a;
        }
};
```

Exercícios

- Crie um classe chamada Ponto:
- Seus atributos são as suas coordenadas x e y
- Implemente um construtor que recebe estes dois parâmetros
- Defina os métodos gets e os sets normalmente
- Escreva um método chamado equals que recebe um outro ponto como argumento retornando true se as coordenadas de ambos forem iguais e false caso contrário.
- Implemente um método chamado distancia que recebe um outro ponto como argumento e calcula a distância entre os dois

Exercícios

```
class Ponto {
private:
    double x, y;

public:
    // Construtor
    Ponto(double x_, double y_) {
        x = x_;
        y = y_;
    }

    // Métodos get
    double getX() const { return x; }
    double getY() const { return y; }

    // Métodos set
    void setX(double x_) { x = x_; }
    void setY(double y_) { y = y_; }
```

Exercícios

```
class Ponto {
private:
    double x, y;

public:
    // Construtor
    Ponto(double x_, double y_) {
        x = x_;
        y = y_;
    }

    // Métodos get
    double getX() const { return x; }
    double getY() const { return y; }

    // Métodos set
    void setX(double x_) { x = x_; }
    void setY(double y_) { y = y_; }

    //Método equals
    bool equals(Ponto outro) {
        return (x == outro.x && y == outro.y);
    }

    //Método distancia
    double distancia(Ponto outro) {
        double dx = x - outro.x;
        double dy = y - outro.y;
        return sqrt(dx * dx + dy * dy);
    }
};
```

Agora, vamos criar 2 pontos na main e testar os métodos..

Exercícios

```
// Exemplo de uso
int main() {
    Ponto p1(2.0, 3.0);
    Ponto p2(5.0, 7.0);

    cout << "Pontos iguais? " << (p1.equals(p2) ? "Sim" : "Não") << endl;
    cout << "Distância entre p1 e p2: " << p1.distancia(p2) << endl;

    return 0;
}
```

Exercícios

1 - Crie uma classe em C++ chamada Relogio para armazenar um horário, composto por hora, minuto e segundo. A classe deve representar esses componentes de horário e deve apresentar os métodos descritos a seguir:

- um método chamado setHora, que deve receber o horário desejado por parâmetro (hora, minuto e segundo);
- um método chamado getHora para retornar o horário atual, através de 3 variáveis passadas por referência;
- um método para avançar o horário para o próximo segundo (lembre-se de atualizar o minuto e a hora, quando for o caso).

Exercícios

```cpp
#include <iostream>

using namespace std;

class Relogio {
private:
    int hora;
    int minuto;
    int segundo;

public:
    // Define o horário desejado
    void setHora(int h, int m, int s) {
        if (h >= 0 && h < 24) hora = h; else hora = 0;
        if (m >= 0 && m < 60) minuto = m; else minuto = 0;
        if (s >= 0 && s < 60) segundo = s; else segundo = 0;
    }

    // Retorna o horário atual por referência
    void getHora(int &h, int &m, int &s) {
        h = hora;
        m = minuto;
        s = segundo;
    }
```

Exercícios

```cpp
#include <iostream>

using namespace std;

class Relogio {
private:
    int hora;
    int minuto;
    int segundo;

public:
    // Define o horário desejado
    void setHora(int h, int m, int s) {
        if (h >= 0 && h < 24) hora = h; else hora = 0;
        if (m >= 0 && m < 60) minuto = m; else minuto = 0;
        if (s >= 0 && s < 60) segundo = s; else segundo = 0;
    }

    // Retorna o horário atual por referência
    void getHora(int &h, int &m, int &s) {
        h = hora;
        m = minuto;
        s = segundo;
    }

    // Avança o relógio em um segundo
    void avancarSegundo() {
        segundo++;
        if (segundo == 60) {
            segundo = 0;
            minuto++;
            if (minuto == 60) {
                minuto = 0;
                hora++;
                if (hora == 24) {
                    hora = 0;
                }
            }
        }
    }
};
```

Exercícios

```
int main() {
    Relogio r;
    r.setHora(23, 59, 59);

    int h, m, s;
    r.getHora(h, m, s);
    cout << "Hora atual: " << h << ":" << m << ":" << s << endl;

    r.avancarSegundo();

    r.getHora(h, m, s);
    cout << "Após 1 segundo: " << h << ":" << m << ":" << s << endl;

    return 0;
}
```

Saída

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

int main() {
    cout << "Olá, mundo!" << endl;
    return 0;
}
```

Saída

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

#define PRECO 1.99

int main()
{
    int pera = 3;
    char qualidade = 'A';
    float peso = 2.5;

    cout << "Existem " << pera << "peras de qualidade " << qualidade
    << "pesando " << peso << "quilos." << endl;

    cout << "O preco por quilo eh R$" << PRECO << ", o total eh R$"
    << peso * PRECO << endl;
}
```

Entrada

```
#include <iostream>
using namespace std;
int main()
{
    int idade;
    cout << "Entre sua idade: ";
    cin >> idade
    cout << "Voce tem " << idade << "anos." << endl;
}
```

Exercícios

1) Definir uma classe que represente um círculo. Esta classe deve possuir métodos Privados para:

- calcular a área do círculo;
- calcular a distância entre os centros de 2 círculos;
- calcular a circunferência do círculo.

E métodos Públicos para:

- definir o raio do círculo, dado um número real;
- aumentar o raio do círculo, dado um percentual de aumento;
- definir o centro do círculo, dada uma posição (X,Y);
- imprimir o valor do raio;
- imprimir o centro do círculo.
- imprimir a área do círculo.

Criar um programa principal para testar a classe.

2) Escreva um programa em C++ que define uma classe “Lampada”.
O estado da lâmpada deve ser representado de forma privada mas deve ser acessível através dos seguintes métodos de instância:

- estaAcesa(),
- estaApagada(),
- acender(),
- apagar().

A classe deve manter o registro de quantas lâmpadas (instâncias/objetos) existem.
O programa deve demostrar que a classe monitora as instâncias; use o destruidor para decrementar o número de instâncias!

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

class Lampada {
private:
    bool acesa;
    static int totalLampadas; // variável estática (compartilhada por todas as instâncias)

public:
    // Construtor
    Lampada() {
        acesa = false; // por padrão, está apagada
        totalLampadas++;
        cout << "Nova lâmpada criada. Total: " << totalLampadas << endl;
    }

    // Destrutor
    ~Lampada() {
        totalLampadas--;
        cout << "Lâmpada destruída. Restantes: " << totalLampadas << endl;
    }
```

```
// Métodos de controle
void acender() {
    acesa = true;
}

void apagar() {
    acesa = false;
}

bool estaAcesa() const {
    return acesa;
}

bool estaApagada() const {
    return !acesa;
}

// Método estático para acessar o total de lâmpadas
static int getTotalLampadas() {
    return totalLampadas;
}
```

```cpp
// Inicialização da variável estática
int Lampada::totalLampadas = 0;

// ==============================
// Programa Principal
// ==============================
int main() {
    cout << "\nCriando lampadas...\n";

    Lampada* l1 = new Lampada();
    Lampada* l2 = new Lampada();
    Lampada l3;

    l1->acender();
    if (l1->estaAcesa())
        cout << "Lâmpada 1 está acesa!\n";

    l3.acender();
    if (!l3.estaApagada())
        cout << "Lâmpada 3 também está acesa!\n";

    cout << "\nTotal de lâmpadas existentes: " << Lampada::getTotalLampadas() << endl;

    // Apagando uma lâmpada
    delete l1;
    cout << "\nApós deletar l1:" << endl;
    cout << "Lâmpadas restantes: " << Lampada::getTotalLampadas() << endl;

    return 0; // Ao sair, l2 e l3 são automaticamente destruídas (se não forem ponteiros)
}
```

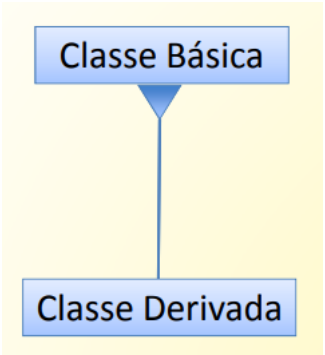
```cpp
#include <iostream>
#include <cmath>
using namespace std;

class Circulo {
private:
    double raio;
    double x, y;

    // Métodos privados
    double calcularArea() const {
        return M_PI * raio * raio;
    }

    double calcularDistancia(const Circulo& outro) const {
        return sqrt(pow(outro.x - x, 2) + pow(outro.y - y, 2));
    }

    double calcularCircunferencia() const {
        return 2 * M_PI * raio;
    }
```

```
public:
    // Construtor
    Circulo() : raio(0.0), x(0.0), y(0.0) {}

    // Métodos públicos
    void definirRaio(double r) {
        if (r >= 0)
            raio = r;
        else
            cout << "O raio deve ser positivo." << endl;
    }

    void aumentarRaio(double percentual) {
        raio *= (1 + percentual / 100.0);
    }

    void definirCentro(double novoX, double novoY) {
        x = novoX;
        y = novoY;
    }

    void imprimirRaio() const {
        cout << "Raio: " << raio << endl;
    }

    void imprimirCentro() const {
        cout << "Centro: (" << x << ", " << y << ")" << endl;
    }

    void imprimirArea() const {
        cout << "Área: " << calcularArea() << endl;
    }

    // Método auxiliar público para testar a distância
    void imprimirDistancia(const Circulo& outro) const {
        cout << "Distância entre centros: " << calcularDistancia(outro) << endl;
    }
};

int main() {
    Circulo c1, c2;

    c1.definirRaio(5);
    c1.definirCentro(2, 3);

    c2.definirRaio(3);
    c2.definirCentro(5, 7);

    c1.imprimirRaio();
    c1.imprimirCentro();
    c1.imprimirArea();

    c1.imprimirDistancia(c2);

    return 0;
}
```

Herança em POO

- A herança permite a criação de uma **nova classe** (classe derivada) a partir de uma **classe existente** (classe de base).

- A classe derivada herda as suas características da classe de base (ou classe pai/mãe), incluindo: Atributos, Métodos
- O que não é herdado: construtores, destrutores, operadores new/atribuição, atributos privados
- Pode ainda adicionar – além dos atributos herdados – outros elementos que lhe são próprios.



Exemplo

```
class Caixa{
    public:
        int altura, largura;
        void Altura(int a) {altura=a;}
        void Largura(int l) {largura=l;}

};

class CaixaColorida : public Caixa{
    public:
        int cor;
        void Cor(int c){cor=c;}
};
```

```
void main(){
    CaixaColorida cc;
    cc.Cor(5);
    cc.Largura(3); //classe herdada
    cc.Altura(50); //classe herdada
}
```

Membros de Classe protected

- Além dos especificadores de acesso **public** e **private**, existe o ***protected*** que funciona como private sob o ponto de vista externo a classe;
- Isto é, atributos ***protected*** são visíveis pelas classes derivadas

```
class A{
    private:
        int a;
    protected:
        int b;
    public:
        int c;
};

class B : public A{
    public:
        int geta(){retrun a;} //ERRO! a não é vizível
        int getb(){retrun b;}
        int getc(){retrun c;}
```

```
void main(){
    A ca;
    B cb;

    ca.a = 1; //ERRO! a não é vizível
    ca.b = 2; //ERRO! b não é vizível
    ca.c = 3;

    cb.a = 4; //ERRO! a não é vizível
    cb.b = 5; //ERRO! b não é vizível
    cb.c = 6;
}
```

Arquivos

- C++ fornece tipos para trabalharmos com leitura e escrita em arquivos
- #include <fstream>

Arquivos

- C++ fornece tipos para trabalharmos com leitura e escrita em arquivos
- #include <fstream>

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

int main () {

 // ofstream - arquivo apenas para saída de dados
 ofstream arq1;

 // função open - abre o arquivo. Cria o arquivo caso ele não exista.
 arq1.open ("nomes.txt");

 // Insere nomes no arquivo (operador "<<")
 arq1 << "Bruno Gomes" << endl;
 arq1 << "Maria Dantas" << endl;

 // função close - fecha o arquivo
 arq1.close();
 return 0;
}
```

Arquivos - ofstream

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

int main () {
 /* Outra forma: arquivo a ser aberto já é colocado diretamente
 após nome da variável arquivo */
 ofstream arq1("nomes.txt");

 //função is_open - testa se o arquivo está realmente aberto
 if (arq1.is_open()) {
 arq1 << "Bruno Gomes" << endl;
 arq1 << "Maria Dantas" << endl;
 arq1.close();
 }

 return 0;
}
```

Arquivos - fgets

```cpp
#include <cstdio>

using namespace std;

int main() {
    char buffer[100];

    printf("Digite uma linha: ");
    if (fgets(buffer, sizeof(buffer), stdin)) {
        printf("Você digitou: %s", buffer);
    } else {
        printf("Erro ao ler a linha.\n");
    }

    return 0;
}
```