



# Programação III

Jordana S. Salamon

[jssalamon@inf.ufes.br](mailto:jssalamon@inf.ufes.br)

[jordanasalamon@gmail.com](mailto:jordanasalamon@gmail.com)

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

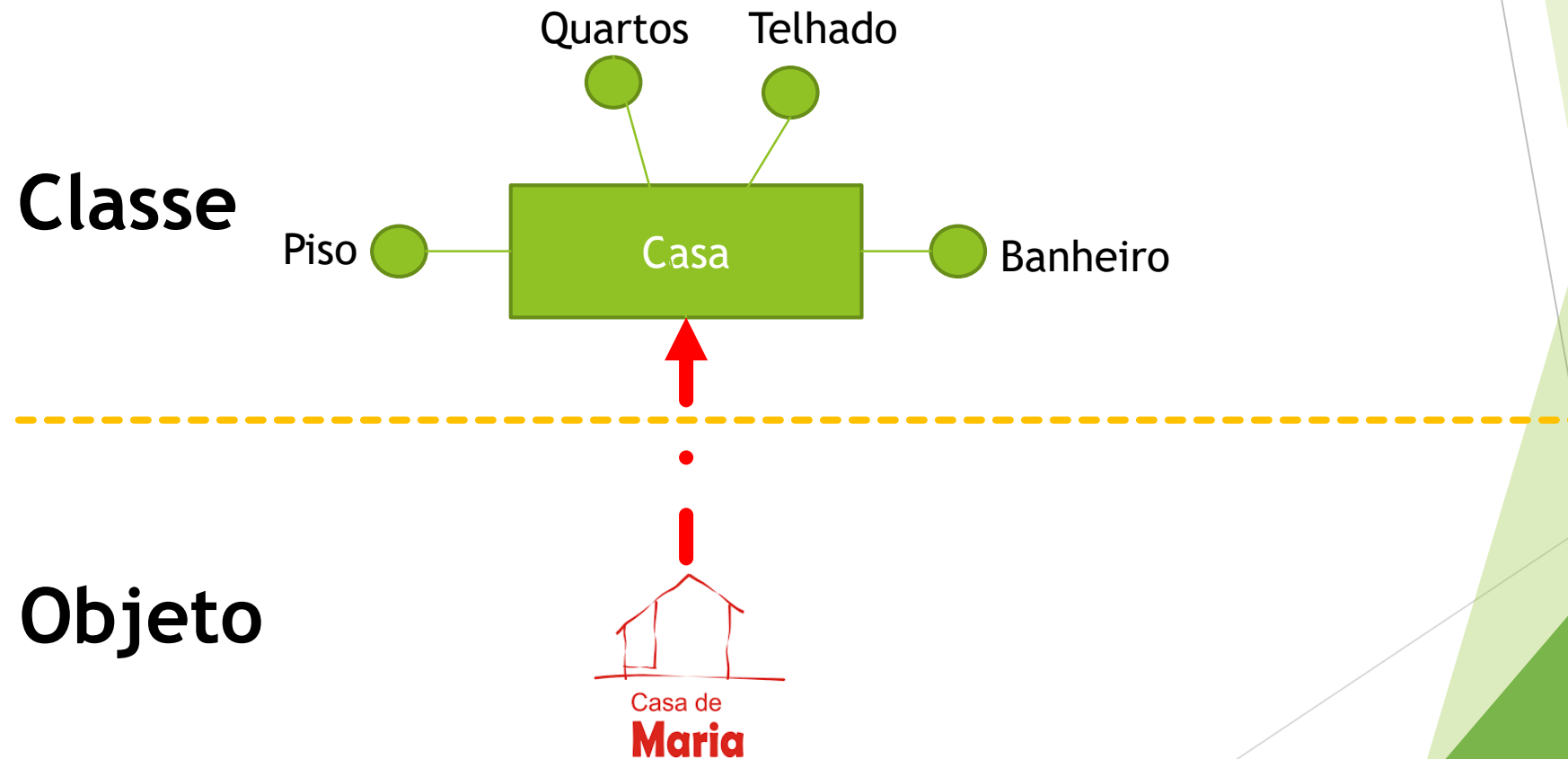
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO



# Revisão Orientação a Objetos

# O que é um Objeto?

- ▶ Um Objeto é instância de uma **Classe**



# Exemplo

## Classe Estudante



Nome

Idade

Atributos

## Métodos:

```
void chorarNota(){
    while(1){
        printf("ponto extra");
    }
}

void pedirProvaEmDupla(){
    while(1){
        printf("Professor, podemos fazer a prova em Dupla?");
    }
}

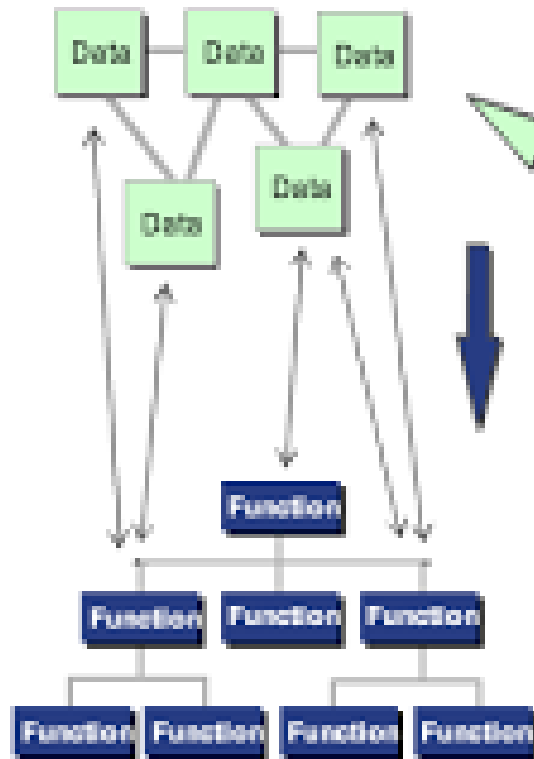
void pedirAdiamentoDoTrabalho(){
    while(1){
        printf("Professor, adia o trabalho, não consegui terminar")
    }
}
```



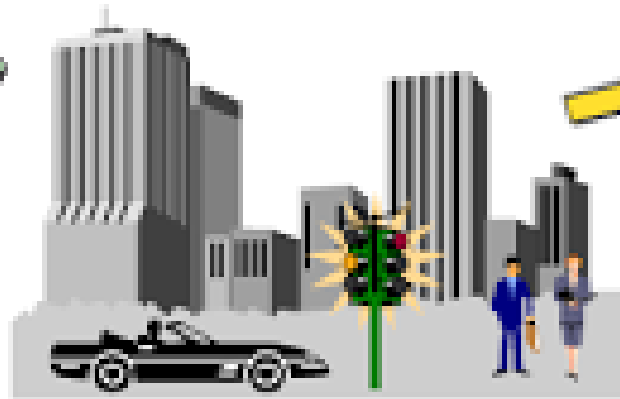
# Comparação Procedural x POO

## Procedural:

Separation of data and functions

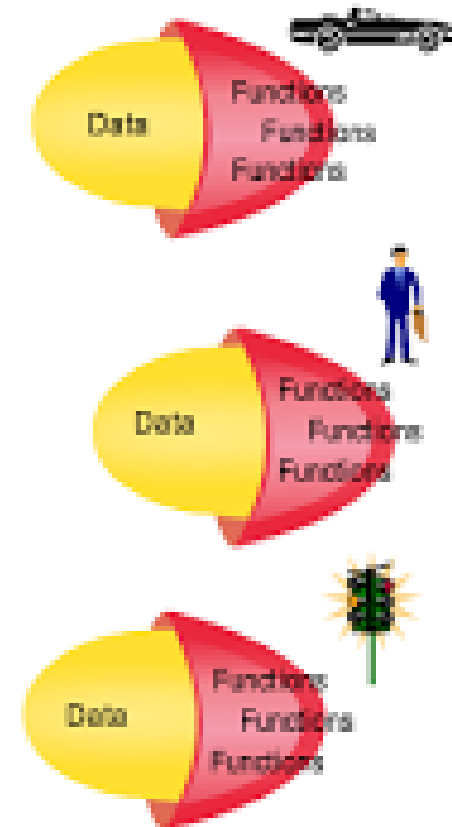


*Real world*



## Object-oriented:

Encapsulation of data and functions



# Linguagem Procedural

```
#include <stdio.h>

struct retangulo{
    float base;
    float altura;
    float area;
};

typedef struct retangulo Retangulo;

float calculaAreaRetangulo(Retangulo r) {
    return r.base * r.altura;
}

int main() {
    Retangulo r;
    r.base = 10;
    r.altura = 10;
    r.area = calculaAreaRetangulo(r);
    printf("Area = %f\n", r.area);

    return 0;
}
```

# Linguagem OO

```
Classe Retangulo{  
    float altura;  
    float base;  
    float area;  
  
    void calculaArea(){  
        area = altura*base;  
    }  
}
```

Retangulo ret1;

ret1 = novo Retangulo();

ret1.altura = 10;

ret1.base = 10;

ret1.calculaArea();

escreva(ret1.area);



10

10

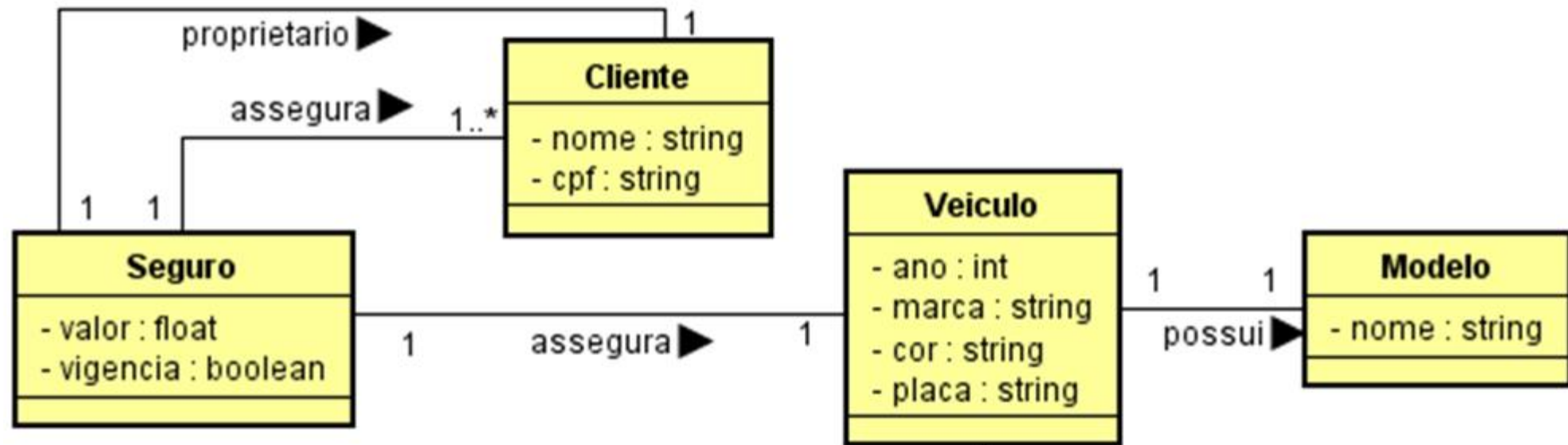
# Exercício

- ▶ Desenvolva um domínio de seguros de carro, modelando as seguintes classes:
  - ▶ Carro: ano, marca, modelo, cor, placa;
  - ▶ Modelo (Carro): nome;
  - ▶ Seguro: carro, cliente, valor, vigência;
  - ▶ Cliente: nome, cpf;





# Resolução





# Introdução a C++

# Mas primeiro...

- ▶ Qual a diferença entre C e C++?
- ▶ O princípio básico da programação estruturada (tal como a da linguagem C) é que um programa pode ser dividido em três partes que se interligam:
  - ▶ **Sequência:** são implementados os passos de processamento necessários para descrever determinada funcionalidade.
  - ▶ **Seleção:** o fluxo a ser percorrido depende de uma escolha. Ex: If, Else, Switch e Case;
  - ▶ **Iteração:** é permito a execução de instruções de forma repetida, onde ao fim de cada execução a condição é reavaliada e enquanto seja verdadeira a execução de parte do programa continua. Ex: While e For.



# O que C++ traz de novo?

- ▶ Em primeiro lugar a mudança de paradigma, agora Orientado a Objetos.
- ▶ C++ faz uso a estrutura definida na linguagem anterior (e.g., tipos primitivos, seleção, interação, ponteiros) e adiciona novos conceitos que aumentam o poder da linguagem.
  - ▶ O benefício de reutilizar a estrutura é não precisar reinventar a roda.
  - ▶ E o problema nisso é a linguagem permitir que em um mesmo programa sejam mesclados os paradigmas, tornando o código confuso e problemático.
- ▶ As linguagens orientadas a objetos (tal como C++) prezam pela **organização, simplificação e reuso de código.**



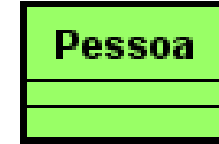
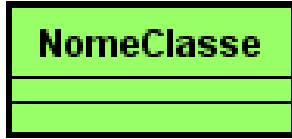
# Sintaxe da Linguagem



nemo

# Definindo Classes

UML



C

```
struct nomeEstrutura{  
    .  
    .  
    .  
};  
typedef struct nomeEstrutura RedefiniçãoNome;
```



```
struct pessoa{  
    .  
    .  
    .  
};  
typedef struct pessoa Pessoa;
```

C++

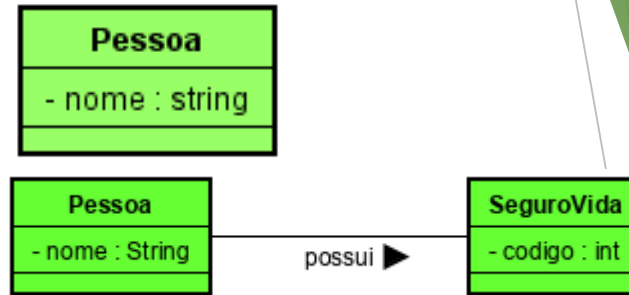
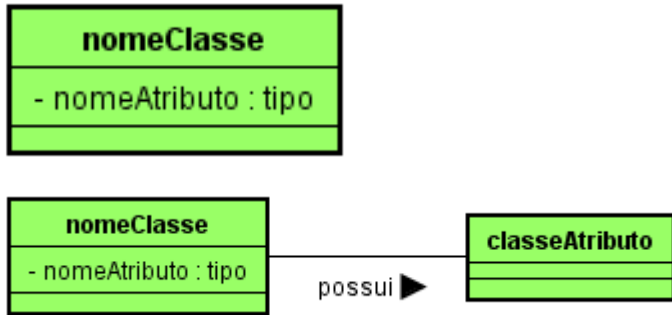
```
class nomeClasse{  
    .  
    .  
    .  
};
```



```
class Pessoa{  
    .  
    .  
    .  
};
```

# Definindo Atributos

UML



C

```
struct estruturaAtributo{
    .
    .
    .
};
typedef struct estruturaAtributo RedefiniçãoNome;

struct nomeEstrutura{
    tipo nomeAtributo;
    EstruturaAtributo estruturaAtributo;
};
typedef struct nomeEstrutura RedefiniçãoNome;
```

```
struct seguroVida{
    int codigo;
};
typedef struct seguroVida SeguroVida;

struct pessoa{
    char* nome;
    SeguroVida seguro;
};
typedef struct pessoa Pessoa;
```

C++

```
class ClasseAtributo{
public:
    .
    .
    .
};

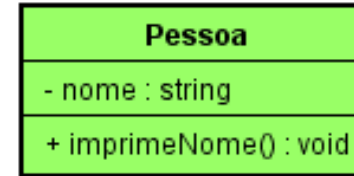
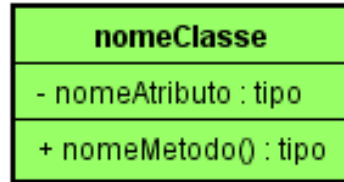
class nomeClasse{
public:
    tipo nomeAtributo;
    ClasseAtributo classeAtributo;
};
```

```
class SeguroVida{
public:
    int codigo;
};

class Pessoa{
public:
    string nome;
    SeguroVida seguro;
};
```

# Definindo Métodos

UML



C

```
struct nomeEstrutura{
    tipo nomeAtributo;
};
typedef struct nomeEstrutura RedefiniçãoNome;

void nomeMetodo(NomeEstrutura x){
    imprime (x.nomeAtributo);
}
```



```
struct pessoa{
    char* nome;
};
typedef struct pessoa Pessoa;

void imprimeNome(Pessoa p){
    printf ("%s", p.nome);
}
```

C++

```
class nomeClasse{
public:
    tipo nomeAtributo;

    void nomeMetodo ()
    {
        imprime (nomeAtributo);
    }
};
```



```
class Pessoa{
public:
    string nome;

    void imprimeNome ()
    {
        imprime (nome);
    }
};
```



# Instanciação

C

```
struct pessoa{
    char* nome;
}
typedef struct pessoa Pessoa;

void imprimeNome(Pessoa p){
    printf ("%s", p.nome);
}

int main()
{
    Pessoa p;
    p.nome = "Gabriel";
    imprimeNome (p);
}
```

C++

```
class Pessoa{
public:
    string nome;

    void imprimeNome ()
    {
        imprime (nome);
    }
};

int main()
    Pessoa p;
    p.nome = "Gabriel";
    p.imprimeNome ();
}
```

```
struct pessoa{
    char* nome;
}
typedef struct pessoa Pessoa;

void imprimeNome(Pessoa * p){
    printf ("%s", p->nome);
}

int main()
{
    Pessoa * p;
    p = (Pessoa*) malloc (sizeof(Pessoa));

    p->nome = "Gabriel";
    imprimeNome (p);
}
```

```
class Pessoa{
public:
    string nome;

    void imprimeNome ()
    {
        imprime (this->nome);
    }
};

int main()
    Pessoa *p;
    p = new Pessoa ();
    p->nome = "Gabriel";
    p->imprimeNome ();
}
```

Por que ponteiros?

1. Economia de Memória RAM.
2. Passagem por referência em funções.
3. Melhora no Tempo de Compilação.
4. O Objeto é criado e destruído pelo usuário, não ocupando a memória todo tempo de execução do programa.
5. Você pode apenas usar o polimorfismo se tiver a referência do objeto.

# Criando e Destruindo Objetos

- ▶ Anteriormente na linguagem C para alocação e liberação de memória dinamicamente em um programa nós utilizávamos a biblioteca `<stdlib.h>`

```
#include <stdlib.h>

int main(){
    Pessoa *p;
    p = (Pessoa *) malloc(sizeof(Pessoa));
    free(p);
    return 0;
}
```

- ▶ Em C++ nós utilizaremos os comandos `new` e `delete` para alocação e liberação de memória dinamicamente.

```
int main()
    Pessoa* p;
    p = new Pessoa();
    delete p;
    return 0;
}
```



# Método Construtor

- ▶ Uma método **construtor** é uma função especial que é executada no momento que criamos novos objetos de uma classe utilizando o **new**;
- ▶ O nome do construtor deve ser exatamente o mesmo que da classe, e não deve ter qualquer tipo de retorno.
- ▶ Ex:

```
class Pessoa{
    string nome;

    Pessoa() {
        .....
        imprime("Estou sendo instanciado");
    }
}

int main()
    Pessoa *p;
    p = new Pessoa();
    p->nome = "Gabriel";
    p->imprimeNome();
}
```



# Método Construtor Parametrizado

- ▶ Uma método construtor pode ser parametrizado, ajudando a inicializar o objeto no momento da instanciação.
- ▶ Ex:

```
class Pessoa{
    string nome;

    Pessoa(string s){
        this->nome = s;
    }
}

int main()
    Pessoa *p;
    p = new Pessoa("Gabriel");
    p->imprimeNome();
}
```

# Método Destruidor

- ▶ Uma método **destruidor** é uma função especial que é executada de forma automática ao final do programa ou pode ser chamada pelo usuário através do comando **delete**;
- ▶ O nome do destruidor deve ser exatamente o mesmo que da classe com um símbolo ‘~’ antes do nome, e não deve ter qualquer tipo de retorno.

▶ Ex:

```
class Pessoa{
    string nome;

    ~Pessoa() {
        .....
        imprime("Estou sendo destruido");
    }
}

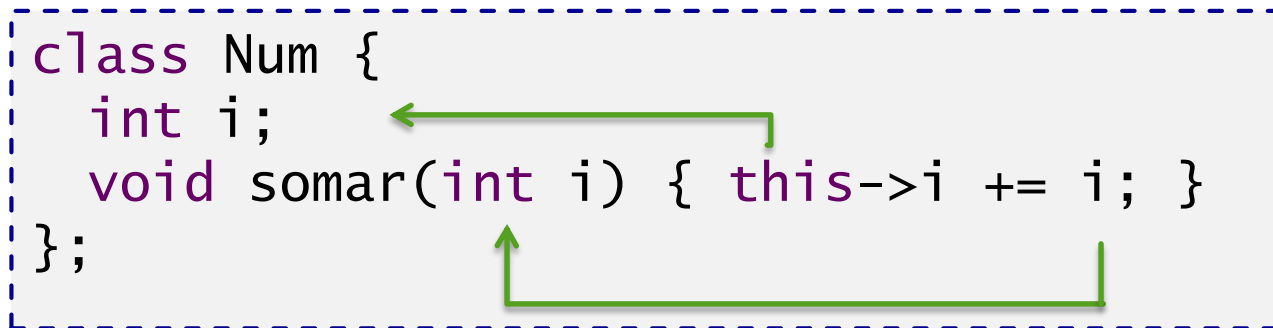
int main()
    Pessoa* p;
    p = new Pessoa();
    p->nome = "Gabriel";
    p->imprimeNome();
    delete p;
}
```



# Ponteiro This

- ▶ A função do ponteiro `this` é explicitar que as variáveis sendo manipuladas dentro dos métodos do objeto são atributos do próprio objeto.
- ▶ Em outras palavras pode ser usado para diferenciar um atributo do objeto de um parâmetro do método:

```
class Num {  
    int i;   
    void somar(int i) { this->i += i; }  
};
```

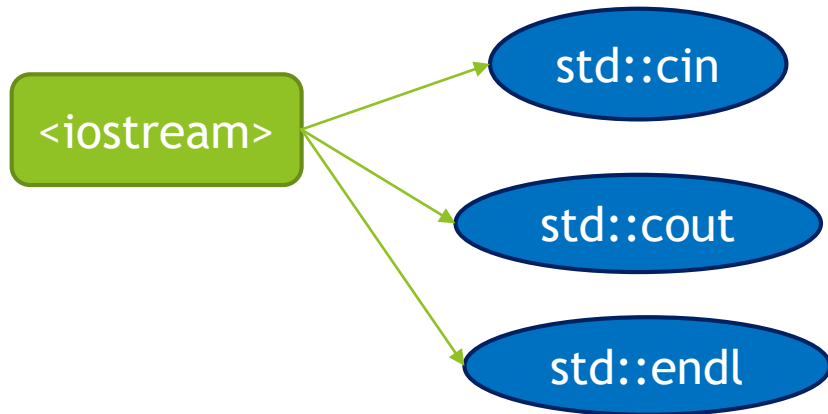


- ▶ Neste caso, o `this` é **necessário!**

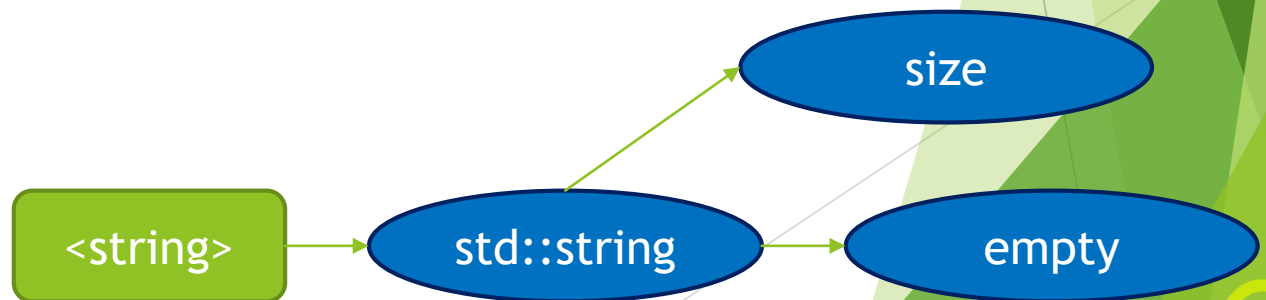
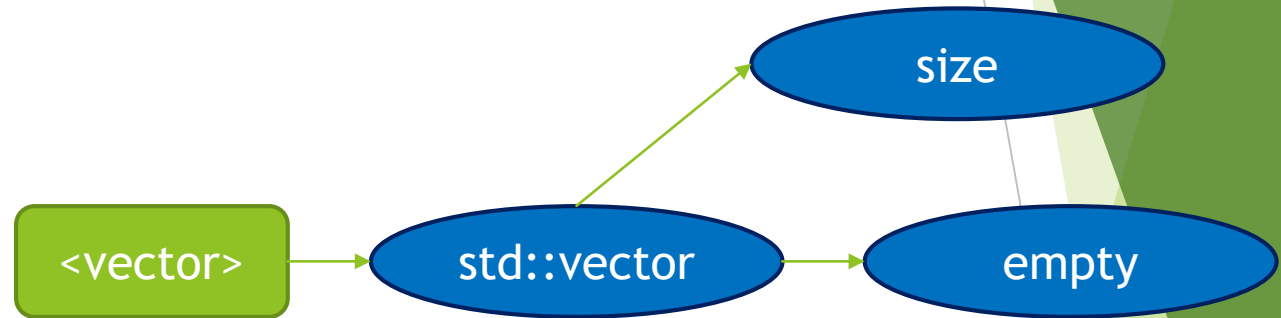
# Bibliotecas Padrões em C++

# Exemplo de Bibliotecas Padrão

## Funções



## Classes





# Entrada e Saída de Dados

- ▶ Anteriormente na linguagem C para leitura e escrita de dados em um programa nós utilizávamos a biblioteca `<stdio.h>`

```
#include <stdio.h>

int main(){
    int x;
    scanf("%d", &x);
    printf("Valor de x: %d", x);

    return 0;
}
```

- ▶ Em C++ nós utilizaremos um abstração chamada *streams* para realizar as operações de entrada e saída de dados. Essas funções estão localizadas na biblioteca `<iostream>`

```
#include <iostream>

int main(){
    int x;
    std::cin >> x;
    std::cout << "Valor de x: " << x <<std::endl;

    return 0;
}
```

# Usando Namespaces

- ▶ Considere a situação que nós temos 2 pessoas com o mesmo nome, os **Namespaces** seriam uma tipo de sobrenome, ou seja, uma informação adicional além do nome para identificar unicamente um elemento.
- ▶ Para dizermos que algo está em um **Namespace**, utilizamos o seguinte padrão:

```
namespace::elemento
```

- ▶ As bibliotecas padrão da linguagem C++ utilizam o namespace **std**. Como por exemplo: **std::cout**.
- ▶ Não vamos estrar no mérito de como criar namespaces neste momento.
- ▶ Vamos apenas abstrai-los e deixar o código mais limpo!



# Usando Namespaces

- ▶ Para abstrair um namespace utilizando a palavra reservada `using`.

```
#include <iostream>

int main() {
    int x;
    std::cin >> x;
    std::cout << "Valor de x: " << x <<std::endl;

    return 0;
}
```

```
#include <iostream>

using namespace std;

int main() {
    int x;
    cin >> x;
    cout << "Valor de x: " << x <<endl;

    return 0;
}
```



# Usando Namespaces

- ▶ Imagine que você está usando duas bibliotecas, X e Y;
  - ▶ Using namespace x;
  - ▶ Using namespace y;
- ▶ Tudo está funcionando perfeitamente.
- ▶ Você usa uma função blablabla() da biblioteca x e uma função blebleble() da biblioteca y.
- ▶ Porém agora você atualizou suas bibliotecas e temos uma nova versão de x, x 2.0!
- ▶ E agora x 2.0 possui uma função blebleble()!
- ▶ Agora você tem um conflito na chamada das funções, porque você está importando duas funções de mesmo nome de bibliotecas diferentes! E pior: ***O código pode chamar a função errada silenciosamente!*** Ao invés de chamar a função blebleble da biblioteca y, pode estar chamado a função da biblioteca x!
- ▶ E isso pode ser um problema ainda mais difícil de detectar se as duas funções blebleble possuírem a mesma lista de parâmetros.

# Manipulando Strings

- ▶ Anteriormente na linguagem C para manipulação de strings em um programa nós utilizávamos a biblioteca `<string.h>`, que nós provia apenas funções.

```
#include <string.h>

int main(){
    int tam;
    char s1[10], *s2;
    s1 = "Gabriel";
    s2 = "Miranda";
    strcpy(s2,s1);
    strcat(s2,s1);
    tam = strlen(s2);
    return 0;
}
```

- ▶ Em C++, a biblioteca `<string>` nos prove a classe `string` que possui um comportamento e funções próprias.

```
#include <string>

int main(){
    int tam;
    string s1, s2;
    s1 = "Gabriel";
    s2 = "Miranda";
    s1 = s2;
    s1 = s1+s2;
    tam = s1.size();
    return 0;
}
```

# Manipulando Containers

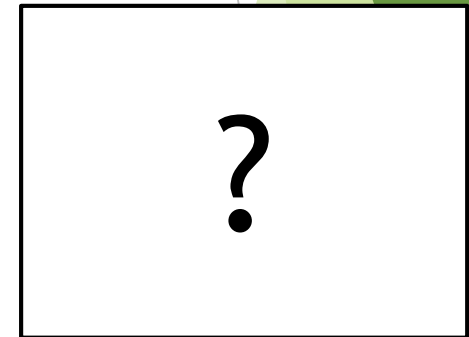
- ▶ Um container é um objeto que armazena uma coleção de outros objetos (seus elementos)
- ▶ O container gerencia o armazenamento de espaço para seus elementos e prove funções para acessar eles.
- ▶ Na linguagem C nós utilizamos containers de diversas maneiras:

```
int main () {  
    int v[10];  
  
    return 0;  
}
```

Tamanho Fixo

```
#include <stdlib.h>  
  
int main () {  
    int *v, n=10;  
    v = (int*) malloc(n*sizeof(int));  
  
    return 0;  
}
```

Tamanho Dinâmico



Tamanho Indeterminado

# Tipos de Containers em C++

- ▶ Existem diversas bibliotecas na linguagem que trabalham com containers:

<bitset>

<map>

<stack>

<set>

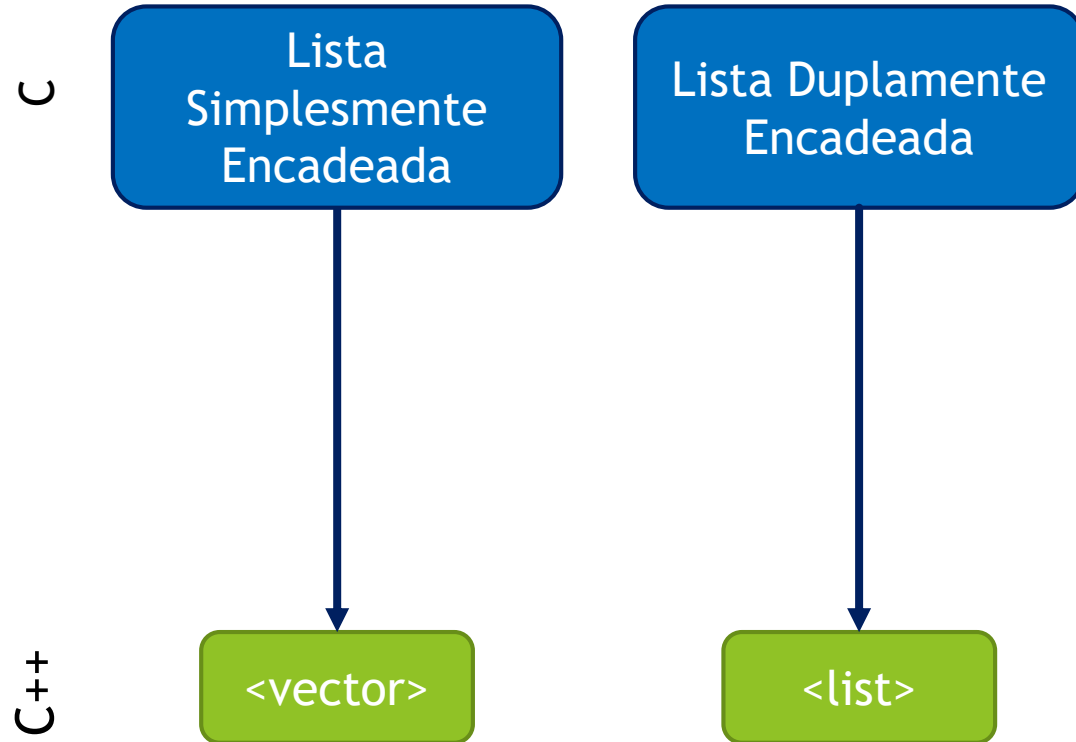
<queue>

<vector>

<list>

<deque>

# Tipos de Containers em C++





# Estrutura <vector>

## ► Estrutura:

```
vector<?> nome;
```

## ► Exemplo:

```
#include <vector>

int main(){
    vector<int> * inteiros;
    vector<float> floats;

    inteiros = new vector<int>();
}
```



# Funções <vector>

- ▶ **Size:** retorna o numero de elemento de um vetor.

```
vector<int> inteiros;  
inteiros.size();
```

- ▶ **Empty:** informa se o vetor está vazio.

```
vector<int> inteiros;  
inteiros.empty();
```

- ▶ **At e Operator[]:** retorna o elemento da posição desejada.

```
vector<int> inteiros;  
inteiros.at(0);  
inteiros[0];
```

- ▶ **Push\_Back:** adiciona um novo elemento ao final do vetor.

```
vector<int> inteiros;  
inteiros.push_back(10);
```



# Funções <vector>

- ▶ **Pop\_Back:** remove o ultimo elemento do vetor.

```
vector<int> inteiros;  
inteiros.pop_back();
```

- ▶ **Insert:** insere um novo elemento antes do elemento de um posição específica. Obs: é necessário passar a referencia para o começo do vector utilizando a função begin.

```
vector<int> inteiros;  
inteiros.insert(inteiros.begin(), 20); //Insere na posicao 0  
inteiros.insert(inteiros.begin()+1, 30); //Insere na posicao 1  
inteiros.insert(inteiros.begin()+2, 40); //Insere na posicao 2
```

- ▶ **Erase:** remove um elemento de uma posição específica dado o inicio do vetor. Neste exemplo será removido o 5º elemento:

```
vector<int> inteiros;  
inteiros.erase(inteiros.begin()+5);
```



# Exercícios

- ▶ 1) Crie uma classe Pessoa, com atributos para nome e telefone;
- ▶ 2) Crie os seguintes métodos para a classe Pessoa:
  - ▶ A) Construtor
  - ▶ B) Destrutor
  - ▶ C) Imprimir nome
  - ▶ D) Imprimir telefone
- ▶ 3) Crie a função main;
- ▶ 4) Dentro da função main crie um vector;
- ▶ 5) Crie pelo menos 3 objetos do tipo Pessoa e insira no vector;
- ▶ 5) Percorra o vector, imprimindo ambas as informações de seus elementos.



That's all Folks!



nemo