

# Programação e Desenvolvimento de Software 2

---

Flavio Figueiredo

<https://github.com/flaviovdf/programacao-2>

# Pessoal

## □ Flavio Figueiredo

### □ Contato

- Prédio do DCC (Anexado ao ICEX) sala 4030
- <http://flaviovdf.github.io> ou  
<http://dcc.ufmg.br/~flaviovdf>
- [flaviovdf@dcc...](mailto:flaviovdf@dcc...)

### □ Dúvidas?

- Qualquer horário assumindo disponibilidade
- Pessoalmente/Moodle/Email

# Sobre a matéria

- PDS2 → Maior foco em desenvolvimento
  - Programação Orientada a Objetos
  - Uso de Estruturas de Dados
  - **Maior foco em desenvolvimento**
- AEDS2 → Maior foco em algoritmos
  - Implementação de Estruturas de Dados
  - Ordenação
  - **Maior foco em algoritmos**

# Programação e Desenvolvimento 2

A ideia é aprender como abstrair o mundo em software

- Entender o problema
- Modelar os dados
- Codificar a solução

# Ementa

- Desenvolvimento de software
- Programação orientada a objetos
- Uso e aplicação de estruturas de dados
- Entendimento da memória
- Boas práticas
  - Testes
  - Programação defensiva

# Avaliação

- 2 provas
  - Cada uma valendo 20 pontos
- 15-20 laboratórios
  - 30 pontos no total
- I Trabalho Prático
  - Valendo 30 pontos

# Trabalho Prático

- Tema da sua escolha
  - Temos uma lista de possíveis
- Grupos de 4/5 alunos
- Código no github
  - Repositório privado
  - Sua tarefa de hoje: criar conta no github

# Desenvolvimento de Software

Saber programar é apenas o passo inicial

- Em pouco tempo conseguimos:
  - if, while, else, for, funções
- Como modelar um programa?
- Como representar um conceito?

# Desenvolvimento de Software

## Exemplos do mundo real

- Como desenvolver um sistema de banco?

# Desenvolvimento de Software

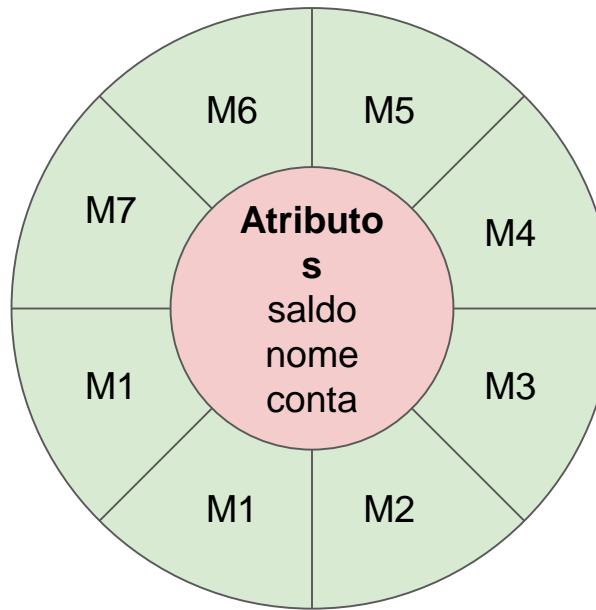
## Exemplos do mundo real

- Como desenvolver um sistema de banco?
- Clientes
- Transações
- Contas
- . . .

# Programação Orientada a Objetos

Uma das formas de modelar o mundo

- Cada entidade do mundo real pode virar um objeto. Será que deve?



# Uso e Aplicação de Estruturas de Dados

A forma que você representa os dados guia seu programa

- Quando é o melhor momento de usar um mapa/dicionário?
- Uma lista?

# Uso e Aplicação de Estruturas de Dados

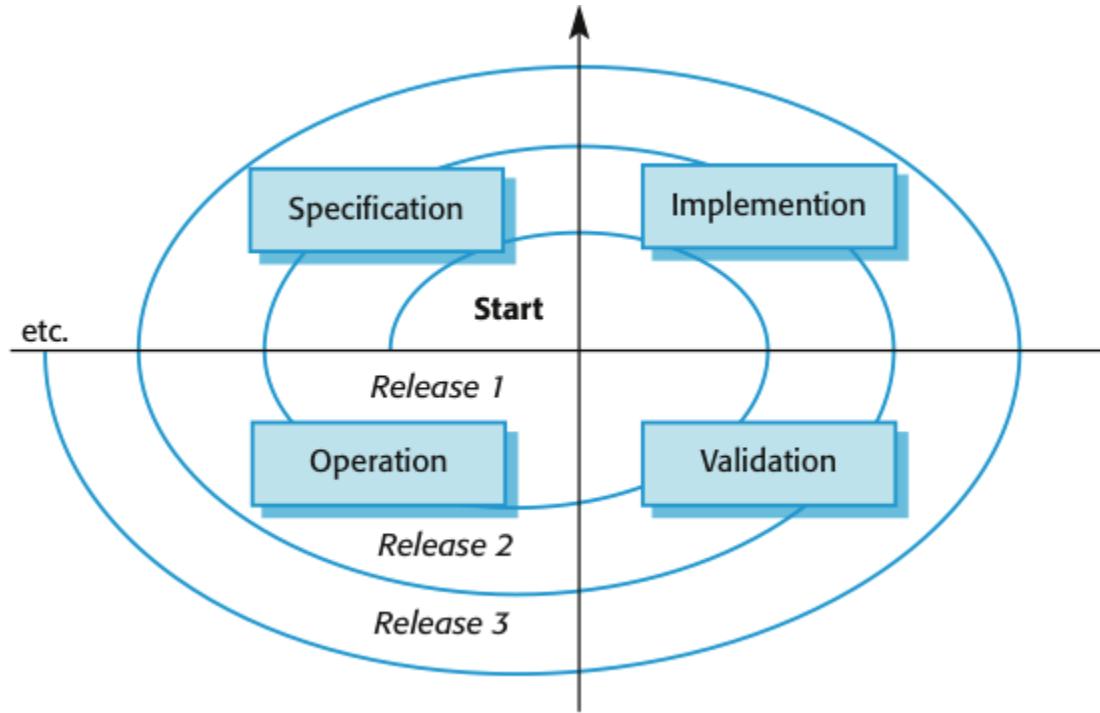
A forma que você representa os dados guia seu programa

- Antigamente AEDS2 focava na implementação de tais estruturas
- Assunto passou para ED
- Vamos prover uma visão top-down

# Boas práticas

Programação é uma atividade social (acreditem ou não)

- PDS2 é o passo 0 para desenvolver software



# C++ Através de Exemplos

---

# Olá Mundo!

```
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

# Olá Mundo!

- Um programa C++ parece com C
- Porém C++ **não** é C
  - São compatíveis

```
#include <iostream>
int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}
```

Incluir biblioteca externa (.h)

Procedimento main

“Atalho” para ‘\n’

Stream da saída padrão

# Compilando

- Usamos o g++

- Similar ao gcc

```
$ g++ hello.cpp -o hello
```

- Saída do programa

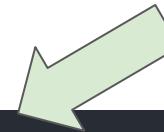
```
$ g++ hello.cpp -o hello
$ ./hello
"Hello World!"
```

# Nesta matéria

- Vale utilizar C++11/14

- Favor não usem C++17

```
$ g++ -std=c++14 -Wall hello.cpp -o hello
```



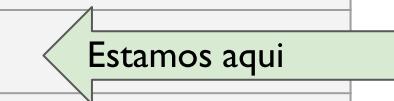
- É comum usar a extensão .cpp

# Padrões de C++

## Pequeno histórico

Ano	Padrão C++	Nome Informal
1998	ISO/IEC 14882:1998	C++98
2003	ISO/IEC 14882:2003	C++03
2011	ISO/IEC 14882:2011	C++11
2014	ISO/IEC 14882:2014	C++14
2017	ISO/IEC 14882:2017	C++20

*Padrão 2017. Ainda não lançado*



# Usando Tipos e STDIN/OUT

```
#include <iomanip>
#include <iostream>
#include <cmath>

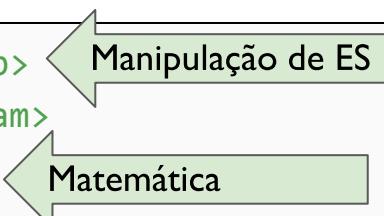
using namespace std;

int main() {
    double pi = 3.1415;
    cout << "Olá DCC :)" ;
    cout << "O valor de pi é? ";
    cout << pi;
    cout << endl;
    cout << "E se eu quiser uma precisão menor? ";
    cout << setprecision(1) << pi;
    cout << endl;
    cout << "Pi ao quadrado com 7 precisão: " << setprecision(7) << pow(pi, 2);
    return 0;
}
```

# Usando Tipos e STDIN/OUT

```
#include <iomanip>           Manipulação de ES
#include <iostream>
#include <cmath>                Matemática
using namespace std;

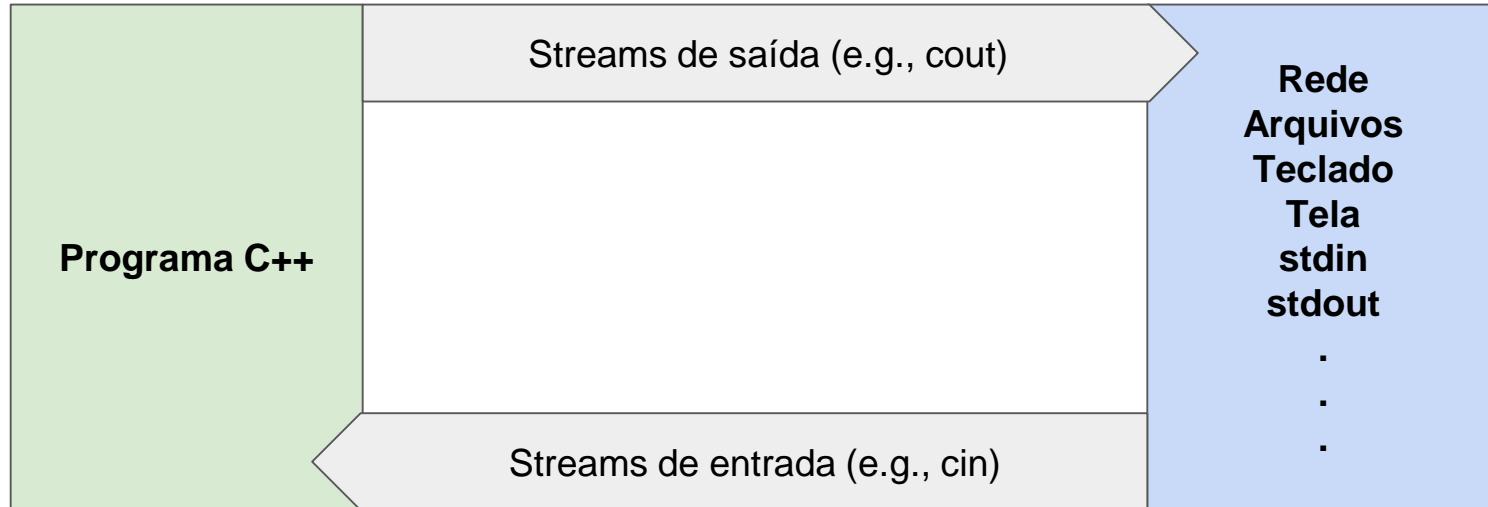
int main() {
    double pi = 3.1415;
    cout << "Olá DCC :)" ;
    cout << "O valor de pi é? ";
    cout << pi;
    cout << endl;
    cout << "E se eu quiser uma precisão menor? ";
    cout << setprecision(1) << pi;
    cout << endl;
    cout << "Pi ao quadrado com 7 precisão: " << setprecision(7) << pow(pi, 2);
    return 0;
}
```



Note como o resultado de  
pow passa pelo filtro  
setprecision

# Streams

- Streams são utilizados para comunicação
- Podemos usar printf também



# Usando funções C

- C++ consegue fazer uso de C
- Vamos tentar manter o curso 100% C++

```
#include <math.h>
#include <stdio.h>

int main() {
    double pi = 3.1415;
    printf("Olá DCC :)\n");
    printf("O valor de pi é? %.2f", pi);
    printf("O valor de pi ao quadrado é? %.2f", pow(pi, 2));
    return 0;
}
```

# Usando STDIN

- Com o cin vamos ler do teclado >>

```
#include <iostream>

int main()  {
    double num1 = 0.0;
    double num2 = 0.0;
    std::cout << "Digite o primeiro número: ";
    std::cin >> num1;
    std::cout << "Digite o segundo número: ";
    std::cin >> num2;
    std::cout << "A divisão de " << num1 << " e " << num2 << " é " \
        << num1/num2 << ".\n";
    return 0;
}
```

Note o std::, sem namespace

# Operadores em C++

Na maioria dos casos, a semântica de C se mantém. Porém...

- Assim como em C, usamos operadores para atuar nos dados:  
+, -, /, \*, >>, >, <
- Porém, o sentido pode mudar dependendo do tipo. Para números >> é shift, para streams é saída.

# Streams em arquivos

## Pouca mudança

```
#include <fstream>
#include <iostream>

using namespace std;

int main() {
    ifstream in("entrada.txt", fstream::in);
    if (!in.is_open()) {
        return 1;
    }
    ofstream out("saída.txt", fstream::out);
    if (!out.is_open()) {
        return 1;
    }
    string line;
    while (getline(in, line)) {
        out << line;
    }
    in.close();
    out.close();
}
```

# Strings

Finalmente! Vamos esquecer o '\0' por um tempo

- C++ tem suporte nativo para strings

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

int main() {
    std::string hello("Olá mundo!\n");
    std::string pds2("Vamos iniciar PDS2\n");
    std::cout << hello;
    std::cout << std::endl;
    std::cout << pds2;

    std::string maisuma = "Mais uma!";
    std::cout << maisuma.size();
    std::cout << std::endl;
    return 0;
}
```

# Diferentes formas de declarar

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

int main() {
    std::string hello1("Olá mundo!\n");
    std::string hello2 = "Olá mundo!\n";
}
```

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

using namespace std;

int main() {
    string hello1("Olá mundo!\n");
    string hello2 = "Olá mundo!\n";
}
```

# Strings

- Suporte nativo ajuda bastante
- Métodos como: .size
  - Tamanho da string
- Note a diferença:
  - ☒ `str.size()` vs `strlen(str)`

# Comparando Strings

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

int main() {
    std::string hello("Olá mundo!\n");
    std::string hello2("Olá mundo!\n");
    if (hello == hello2) {
        std::cout << "c++ faz overload do == para strings!!!!.\n";
    }
    if (hello.compare(hello2) == 0) {
        std::cout << "Strings iguais.\n";
    }
    return 0;
}
```

*Note o overload do operador ==. Comodidade.*

*Mesma coisa de antes*

# Vetores

```
#include <iostream>

int main() {
    int n = 0;
    std::cout << "Digite o número de elementos: ";
    std::cin >> n;

    int dados[n];
    for (int i = 0; i < n; i++) {
        std::cout << "Digite o " << i+1 << "-ésimo número: ";
        std::cin >> dados[i];
    }

    int soma = 0;
    for (int i = 0; i < n; i++) {
        soma += dados[i];
    }
    std::cout << "A soma foi: " << soma << std::endl;
}
```

# Saída

- Sem muita surpresa
- Comandos de repetição estilo C
- Porém, podemos incrementar.

```
$ g++ -Wall -std=c++14 acumulador.cpp -o acumulador
$ ./acumulador
Digite o número de elementos: 3
Digite o 1-ésimo número: 2
Digite o 2-ésimo número: 1
Digite o 3-ésimo número: 6
A soma foi: 9
```

# Vectors

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> dados = {};
    int v = 0;
    int i = 0;
    while (v >= 0) {
        std::cout << "Digite o " << i+1 << "-ésimo número (-1 para terminar): ";
        std::cin >> v;
        if (v < 0) break;
        dados.push_back(v);
    }

    for (int& x : dados)
        x *= 2;
    for (int x : dados)
        std::cout << x << std::endl;
}
```

Vetor redimensionável, uma lista com array por baixo.

Nova forma de iterar

# Nova saída

- Duplicamos o valor dos elementos
- Como?

```
$ g++ -Wall -std=c++14 acumulador2.cpp -o acumulador2
$ ./acumulador2
Digite o 1-ésimo número (-1 para terminar): 3
Digite o 1-ésimo número (-1 para terminar): 4
Digite o 1-ésimo número (-1 para terminar): 7
Digite o 1-ésimo número (-1 para terminar): 8
Digite o 1-ésimo número (-1 para terminar): -1
6
8
14
16
```

# Qual a saída em cada caso?

## □ Laço clássico

```
std::vector<int> dados = {0, 7, 8, 1, 3};  
for (int i = 0; i < dados.size(); i++)  
    std::cout << dados[i];
```

## □ Laço compacto

```
for (int x : dados)  
    std::cout << x;
```

## □ Laço para a referência

```
for (int &x : dados)  
    x *= 2;
```

# Exemplo &

<https://goo.gl/MXw83D>

```
#include <iostream>

int& function(int& f) {
    f=f+3;
    return f;
}

int main() {
    int x = 7;
    int y;
    y = function(x);
    std::cout << "Input: " << x << std::endl;
    std::cout << "Output:" << y << std::endl;
    x++;
    y--;
    std::cout << "X: " << x << std::endl;
    std::cout << "Y:" << y << std::endl;
    return 0;
}
```

# Até agora

## Apresentação inicial da linguagem

- Todo o curso vai ser focado em exemplos
- Vamos explorar melhor os conceitos
  - Exemplos de hoje são motivadores iniciais
- Não é um curso de linguagem!
  - Não podemos focar nos detalhes de C++
  - C++ é uma ferramenta para nosso curso

# Bibliografia

**Clean Code: A Handbook of Agile Software Craftsmanship.**

Robert C. Martin.

Prentice Hall, 2008.

**Code Complete: A Practical Handbook of Software Construction.**

Steve McConnell.

Microsoft Press, 2004. 2nd Edition.

**Effective C++: 55 Specific Ways to Improve Your Programs and Designs.**

Scott Meyers.

Addison-Wesley Professional, 2005. 3rd Edition.

**A Tour of C++.**

Bjarne Stroustrup.

Addison-Wesley Professional, 2013. 1st Edition.

# Por fim

- Criar conta no github
- Configurar um ambiente C++
- Escolher uma IDE. Sugerimos:
  - Visual code studio
  - Linux subsystem for windows
  - Tutoriais de configuração em breve

# Programação e Desenvolvimento de Software 2

## Armazenamento de dados em memória

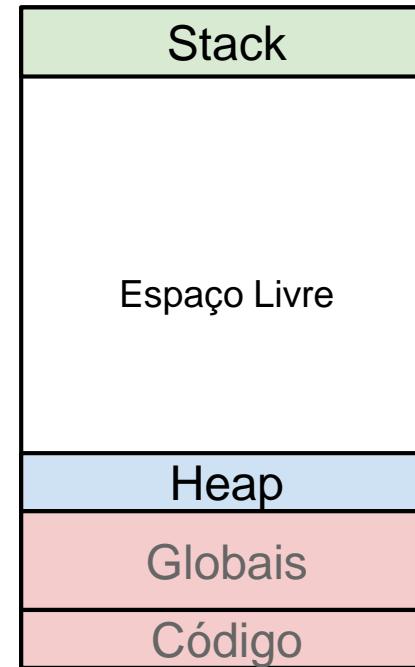
---

Flavio Figueiredo

<https://github.com/flaviovdf/programacao-2>

# Um programa na memória

- Podemos representar um programa em regiões como:
  - Pilha (Stack)
  - Heap
  - Código
  - Globais



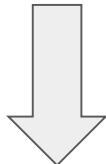
# Compilador

```
$ g++ hello.cpp -o hello
```

Programa

# Compilador

```
$ g++ hello.cpp -o hello
```

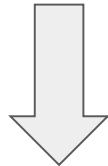


```
0000000000008a5 <main>:  
 8a5: 55                      push    %rbp  
 8a6: 48 89 e5                mov     %rsp,%rbp  
 8a9: 48 8d 35 15 01 00 00    lea     0x115(%rip),%rsi      # 9c5  
<_ZStL19piecewise_construct+0x1>  
 8b0: 48 8d 3d a9 07 20 00    lea     0x2007a9(%rip),%rdi      # 201060  
<_ZSt4cout@@GLIBCXX_3.4>  
 8b7: e8 d4 fe ff ff          callq   790  
<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>  
 8bc: 48 89 c2                mov     %rax,%rdx  
 8bf: 48 8b 05 0a 07 20 00    mov     0x20070a(%rip),%rax      # 200fd0  
<_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_@GLIBCXX_3.4>  
 8c6: 48 89 c6                mov     %rax,%rsi  
 8c9: 48 89 d7                mov     %rdx,%rdi  
 8cc: e8 cf fe ff ff          callq   7a0 <_ZNStolsEPFRSoS_E@plt>  
 8d1: b8 00 00 00 00          mov     $0x0,%eax  
 8d6: 5d                      pop     %rbp  
 8d7: c3                      retq
```

Programa

# Compilador

```
$ g++ hello.cpp -o hello
```

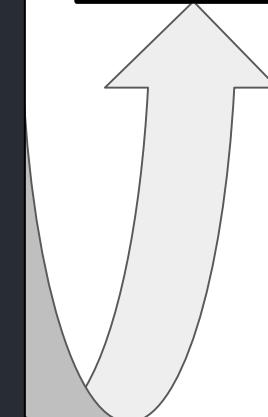


```
0000000000008a5 <main>:  
 8a5: 55                      push   %rbp  
 8a6: 48 89 e5                mov    %rsp,%rbp  
 8a9: 48 8d 35 15 01 00 00    lea    0x115(%rip),%rsi      # 9c5  
<_ZStL19piecewise_construct+0x1>  
 8b0: 48 8d 3d a9 07 20 00    lea    0x2007a9(%rip),%rdi      # 201060  
<_ZSt4cout@@GLIBCXX_3.4>  
 8b7: e8 d4 fe ff ff          callq  790  
<_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>  
 8bc: 48 89 c2                mov    %rax,%rdx  
 8bf: 48 8b 05 0a 07 20 00    mov    0x20070a(%rip),%rax      # 200fd0  
<_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_@GLIBCXX_3.4>  
 8c6: 48 89 c6                mov    %rax,%rsi  
 8c9: 48 89 d7                mov    %rdx,%rdi  
 8cc: e8 cf fe ff ff          callq  7a0 <_ZNSolsEPFRSoS_E@plt>  
 8d1: b8 00 00 00 00          mov    $0x0,%eax  
 8d6: 5d                      pop    %rbp  
 8d7: c3                      retq
```

Programa

Globais

Código



# Execução de um Programa

Ciclo fetch/execute -- Assunto de OC/SO

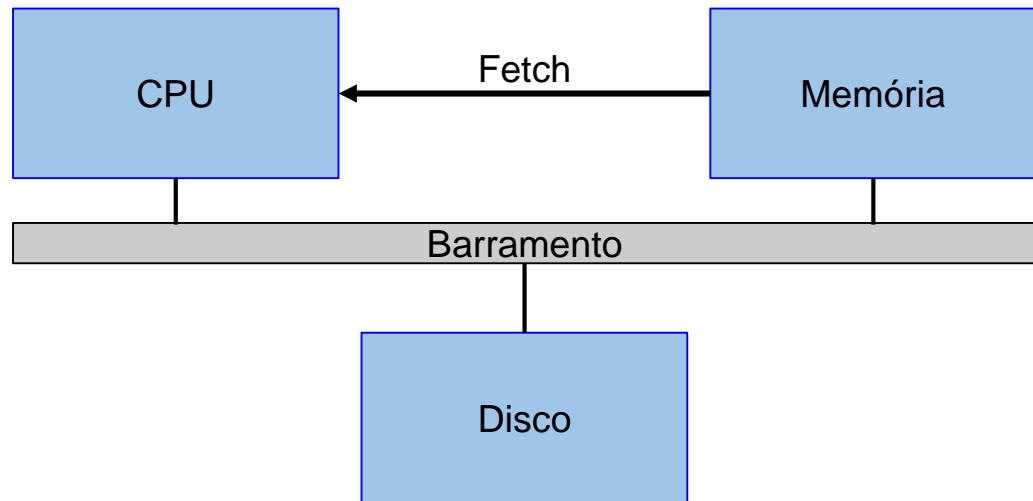
## □ Fetch

- Recupera instrução
- e.g., movsd

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y;
    return 0;
}
```

- CPU vai executar movsd



# Execução de um Programa

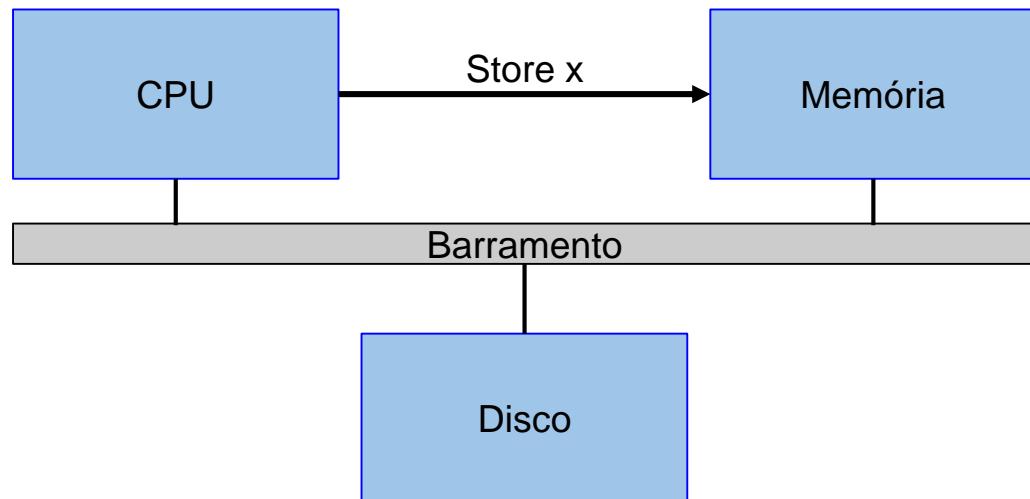
Ciclo fetch/execute -- Assunto de OC/SO

- A instrução faz store

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y;
    return 0;
}
```

- x agora na memória



# Execução de um Programa

Ciclo fetch/execute -- Assunto de OC/SO

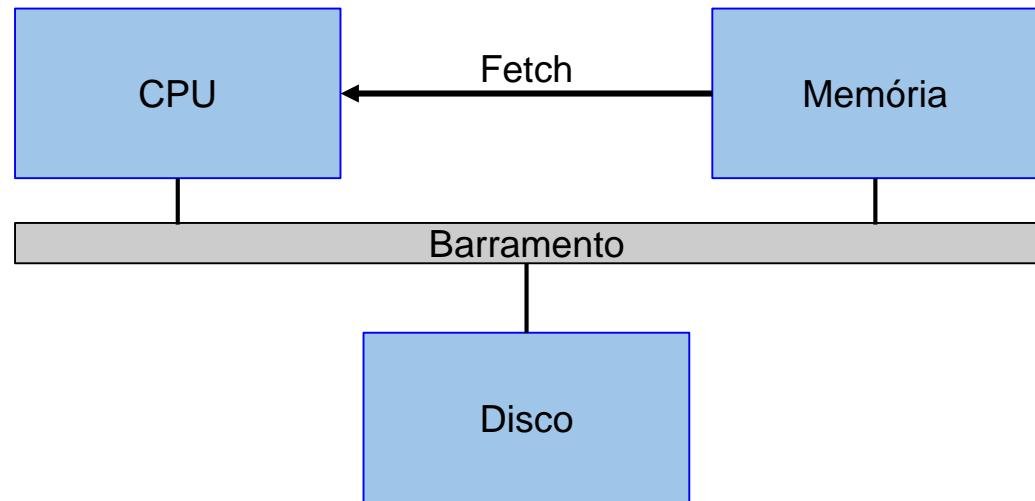
- Fetch

- Recupera instrução
- e.g., movsd

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y;
    return 0;
}
```

- CPU vai executar movsd



# Execução de um Programa

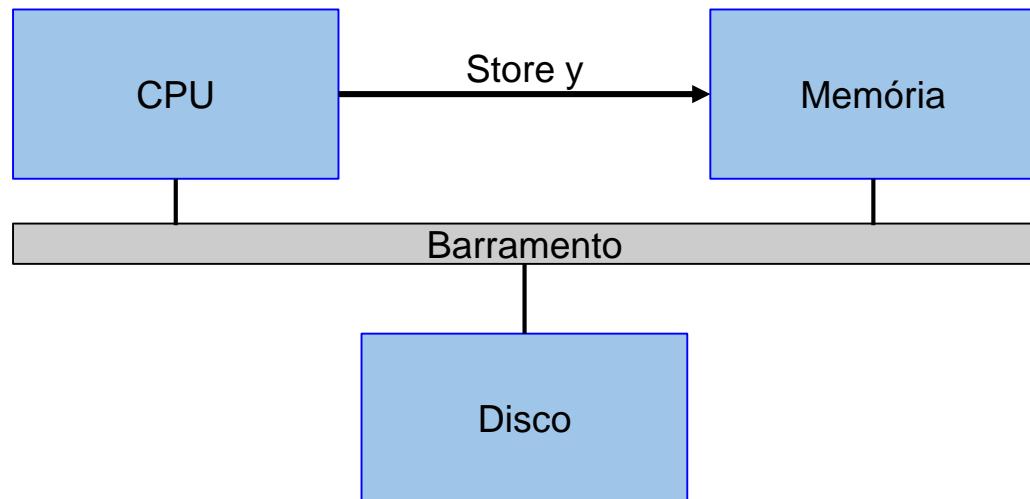
Ciclo fetch/execute -- Assunto de OC/SO

- A instrução faz store

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y;
    return 0;
}
```

- y agora na memória



# Execução de um Programa

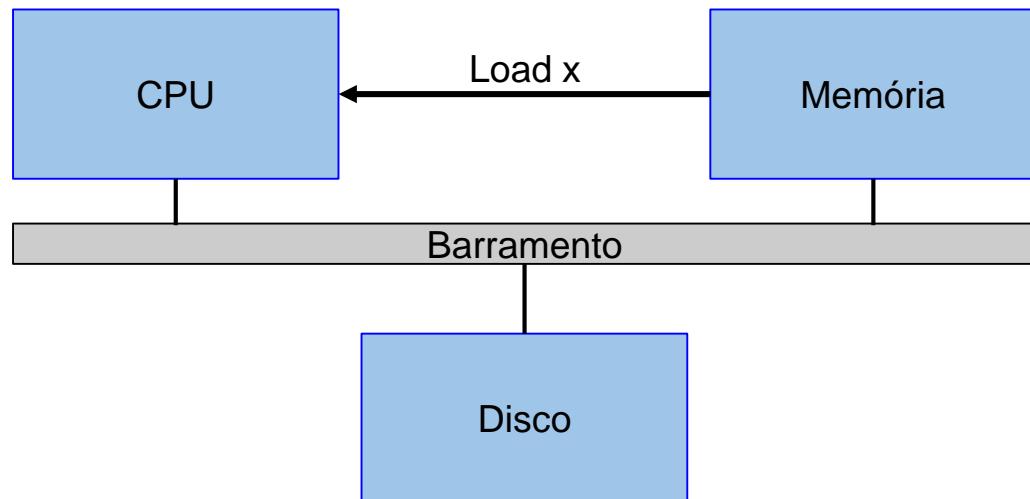
Ciclo fetch/execute -- Assunto de OC/SO

## Load

□ Recuperamos  
x/y pois é  
necessário

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y; ←
    return 0;
}
```



# Execução de um Programa

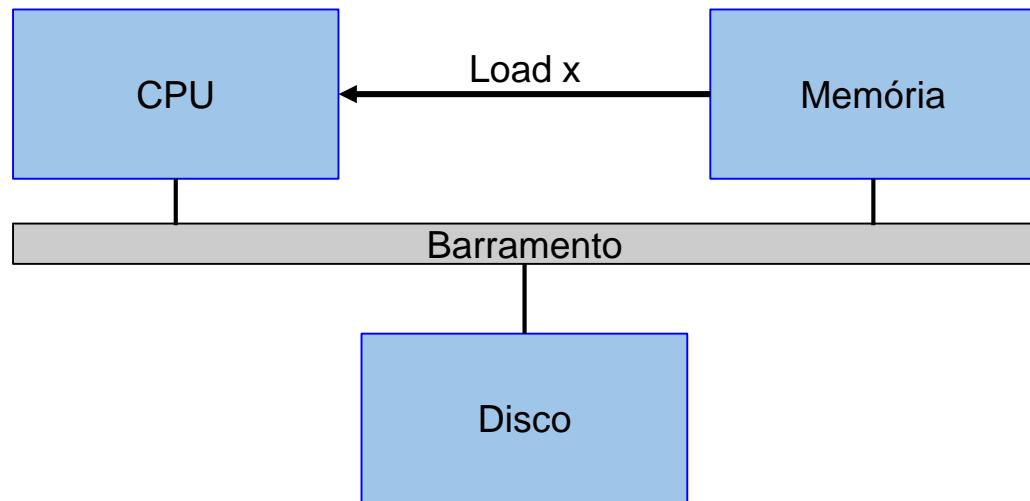
Ciclo fetch/execute -- Assunto de OC/SO

## Load

□ Recuperamos  
x/y pois é  
necessário

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y; ←
    return 0;
}
```



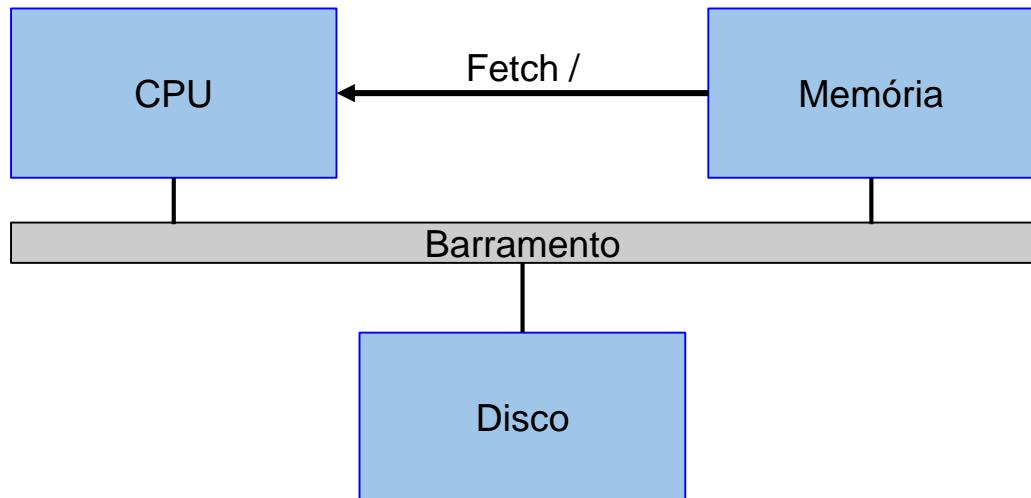
# Execução de um Programa

Ciclo fetch/execute -- Assunto de OC/SO

- Fazemos fetch de divsd (divisão)

```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y;
    return 0;
}
```



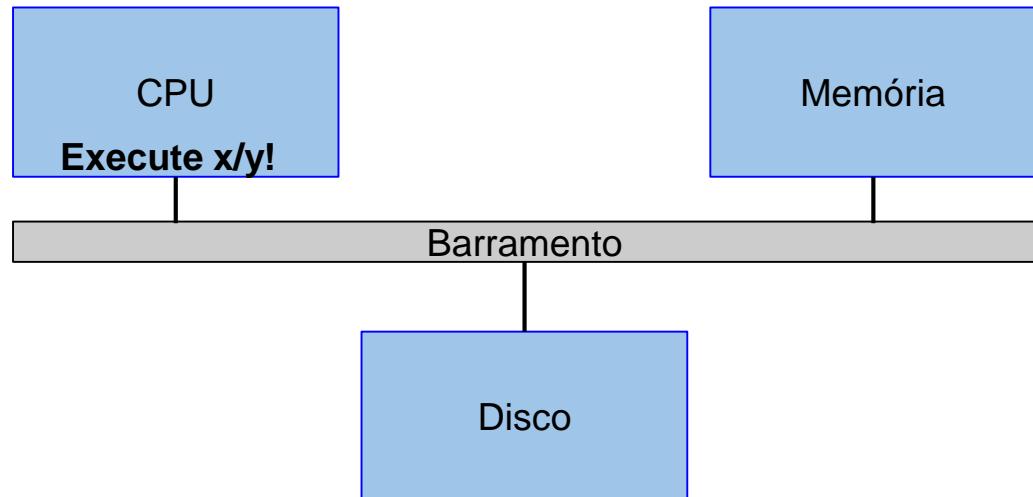
# Execução de um Programa

Ciclo fetch/execute -- Assunto de OC/SO

- Executamos  
 $x/y$
- Fim!

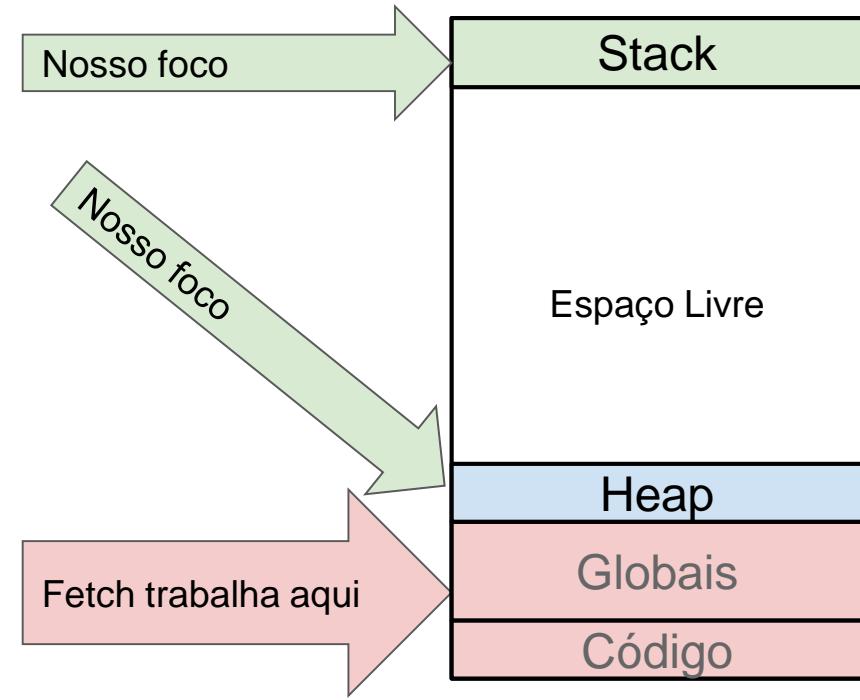
```
#include <iostream>

int main() {
    double x = 71.1;
    double y = 0.01;
    std::cout << x / y; ←
    return 0;
}
```



# Voltando para a memória

- Vamos focar no Stack/Heap
- Onde moram “os dados”
- A parte de código não muda muito de uma linguagem para outra (compilador)



# Compilando

```
$ g++ programa.cpp -o programa
```

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```

nome	ender.	valor
	0x0048	
	0x0044	
	0x0040	
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Compilando

```
$ g++ programa.cpp -o programa
```

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



nome	ender.	valor
	0x0048	
	0x0044	
	0x0040	
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

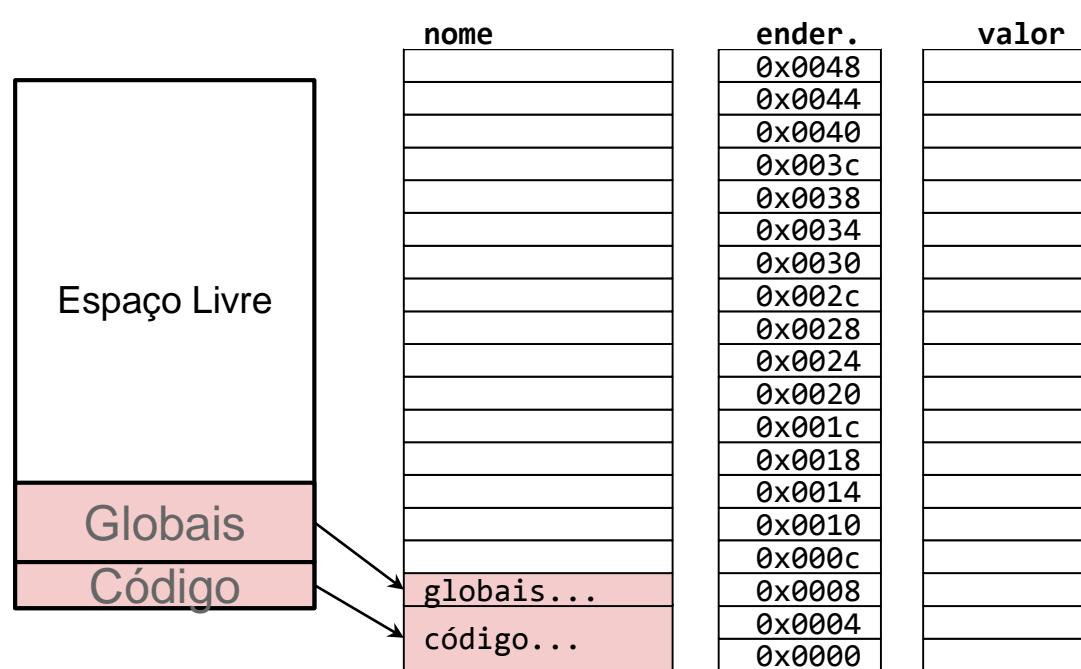
# Dando início

```
$ ./programa
```

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



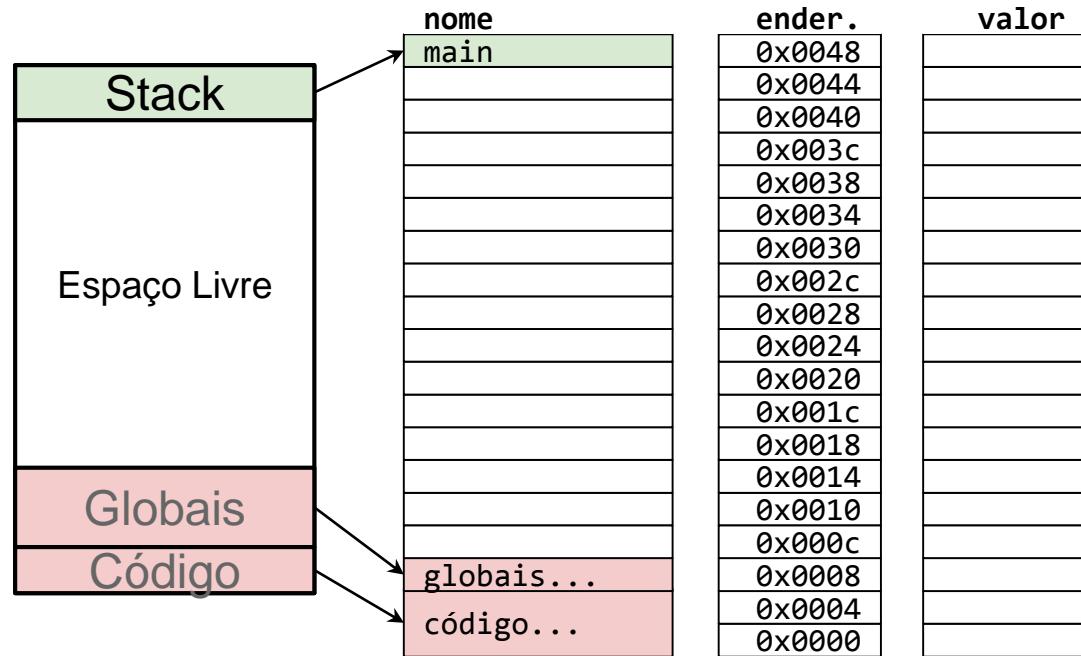
# Stack

Colocamos o main na pilha

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



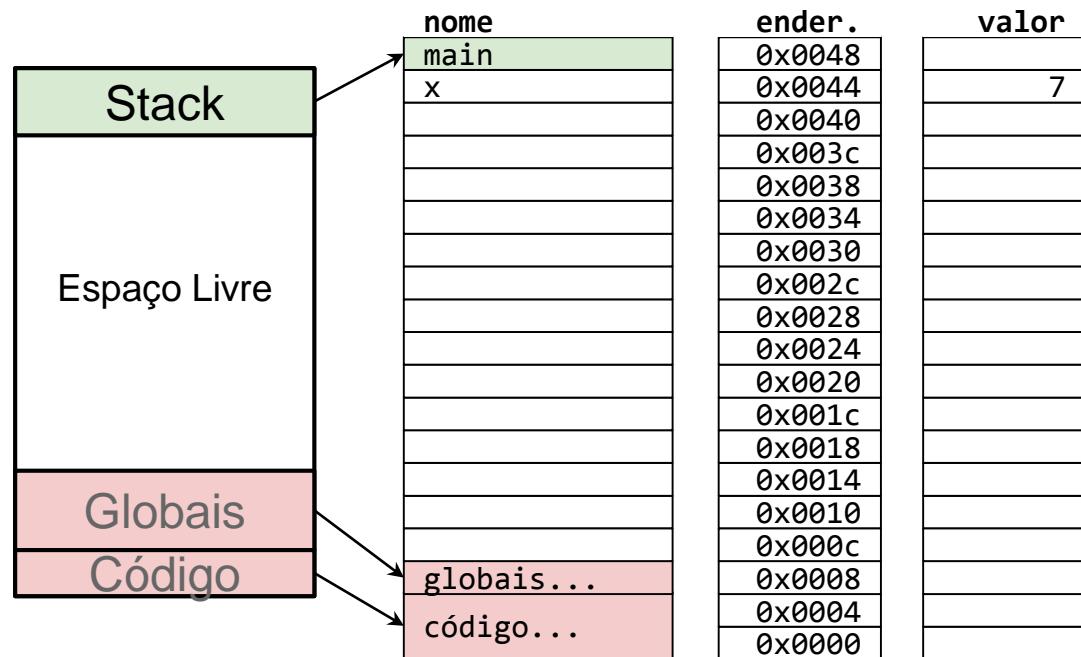
# Stack

x = 7

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7; →
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



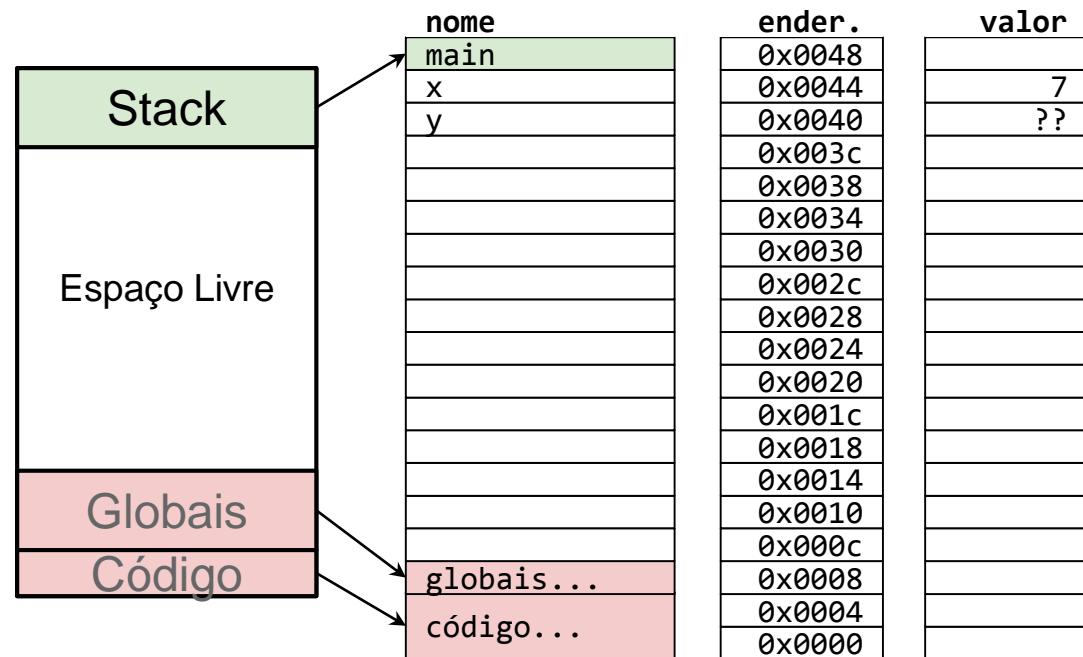
# Stack

y = lixo

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



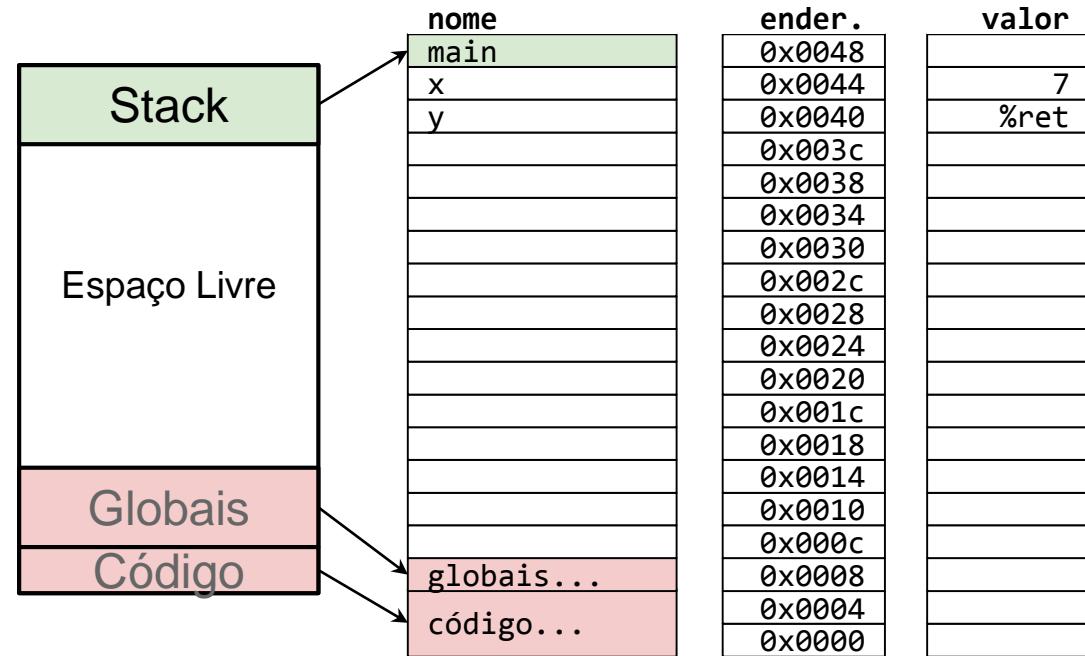
# Stack

y = retorno de function. Representado por %ret

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



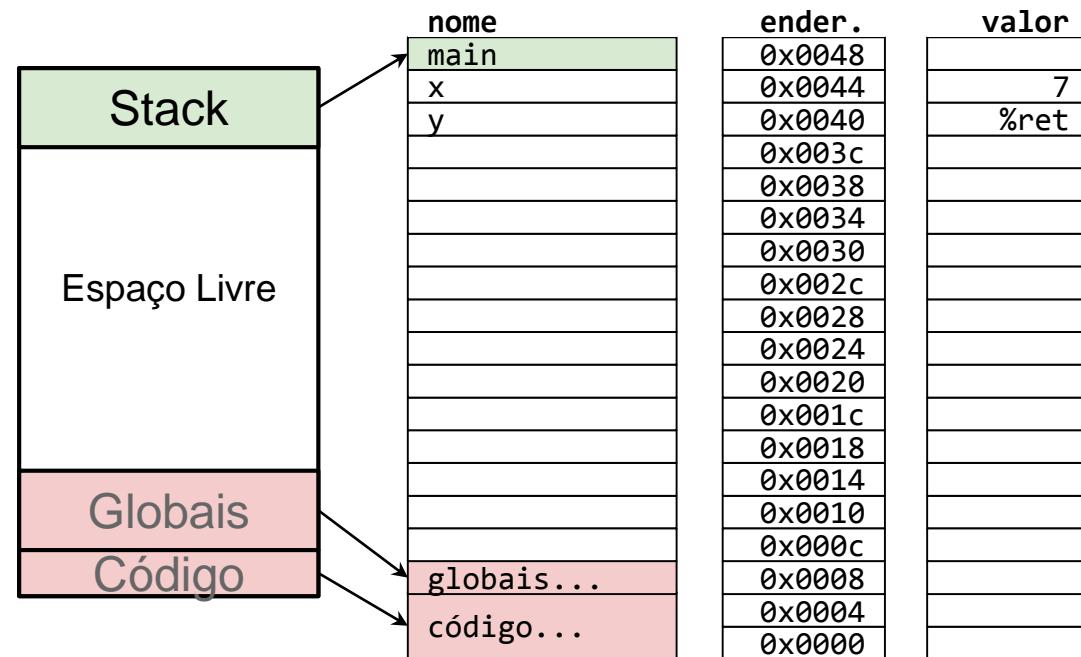
# Stack

y = retorno de function. Representado por %ret

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



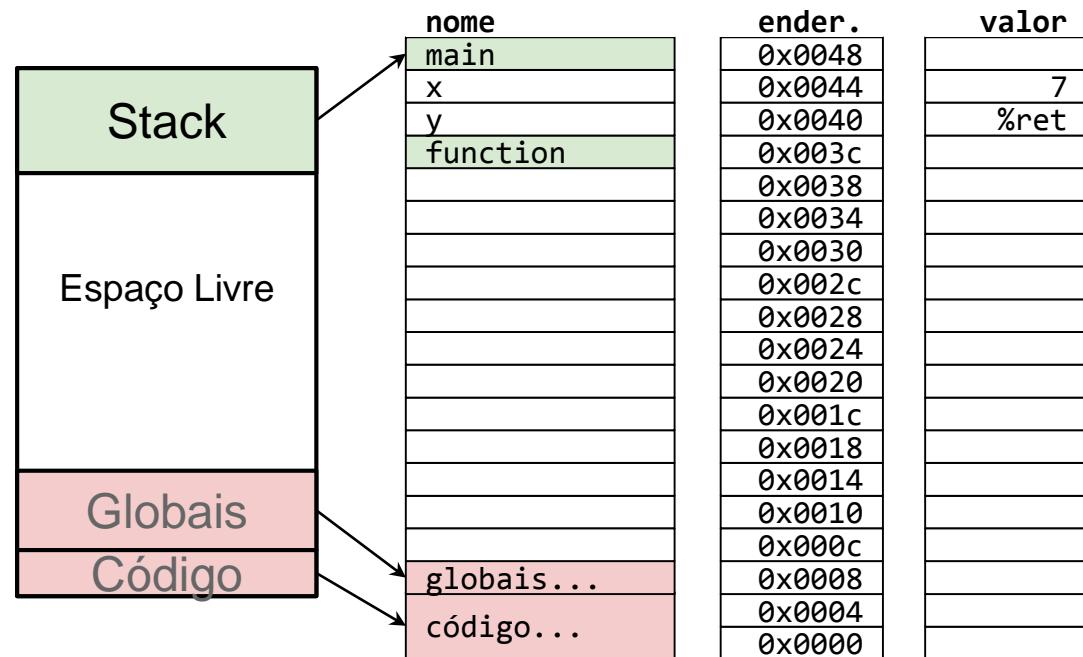
# Stack

chamamos a função `function`

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



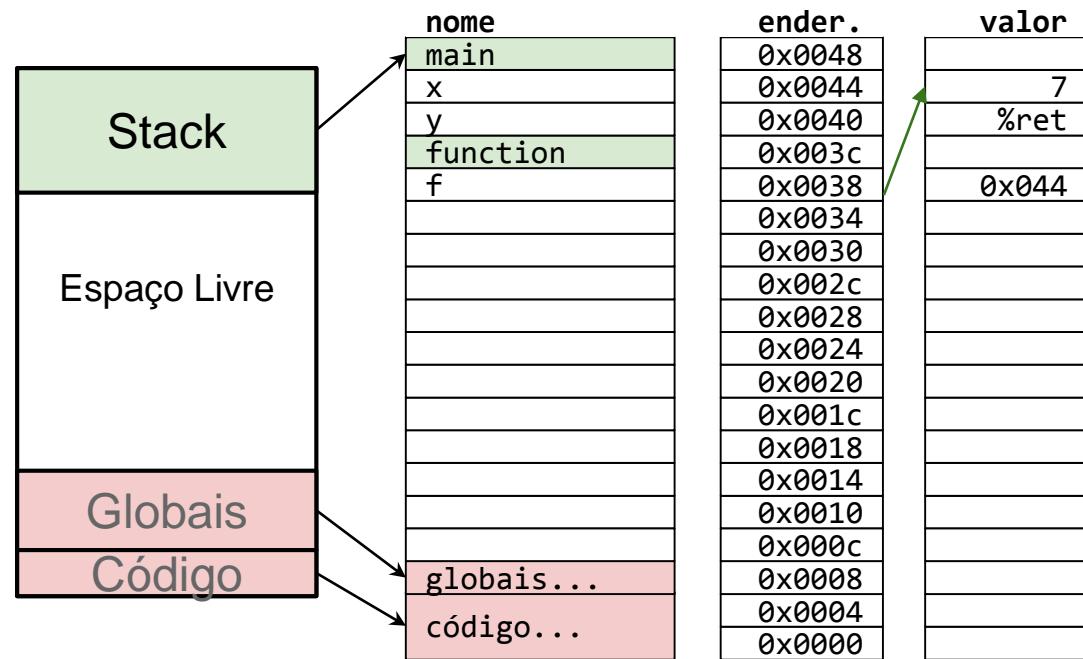
# Stack

empilhamos f. & é uma referência

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



# Stack

empilhamos um ponteiro para f.

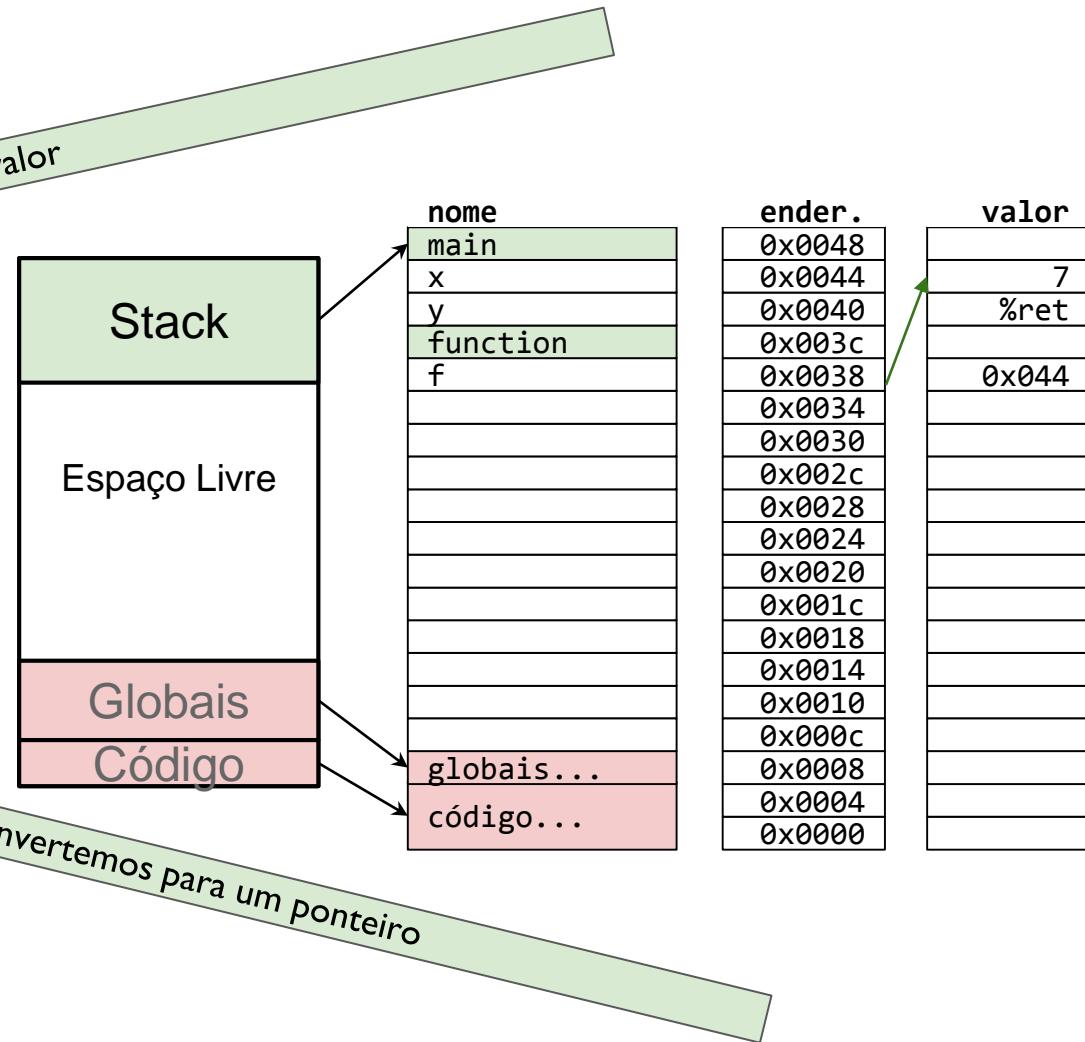
```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```

Usando \* acessamos o valor

Usando & convertemos para um ponteiro



# Stack

atualizamos o valor

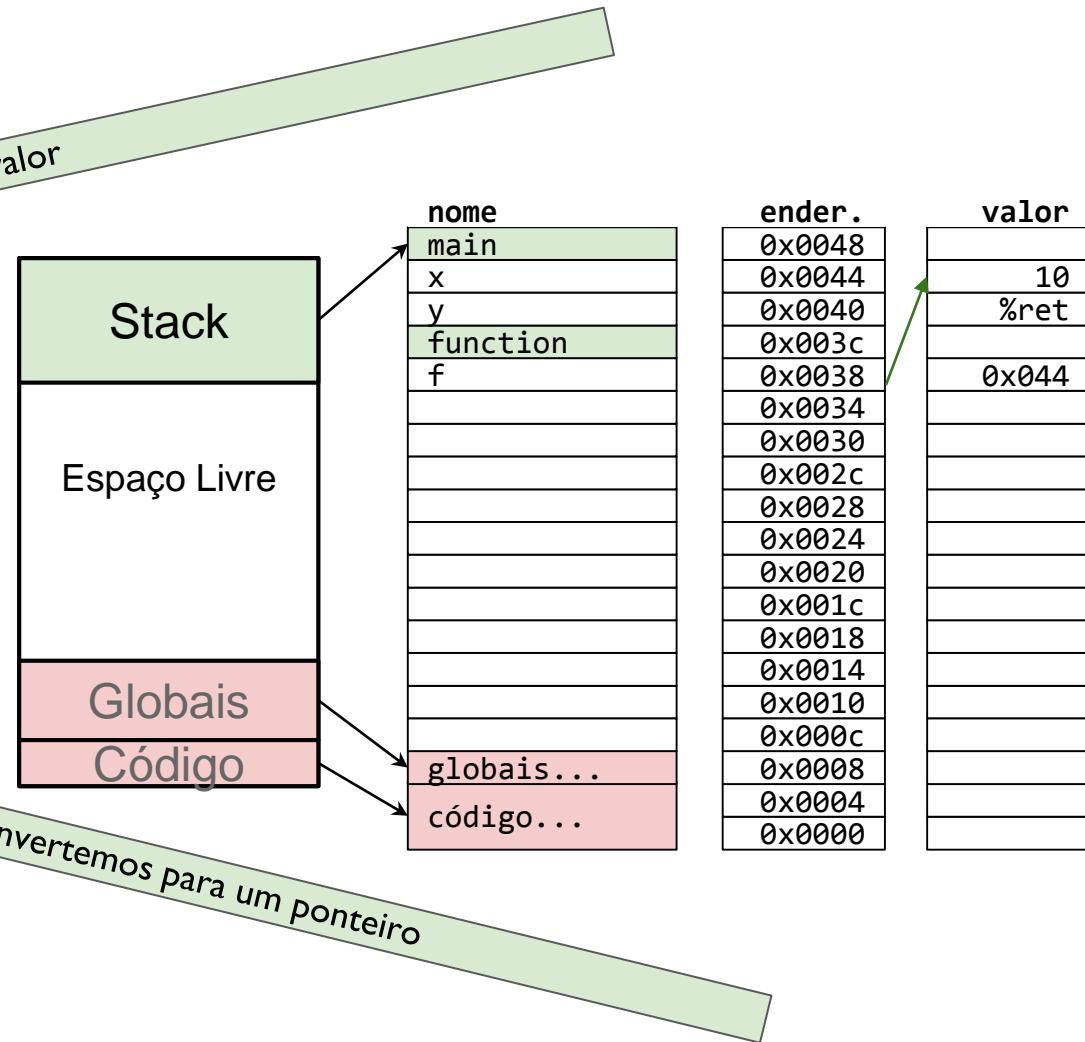
```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```

Usando \* acessamos o valor

Usando & convertemos para um ponteiro

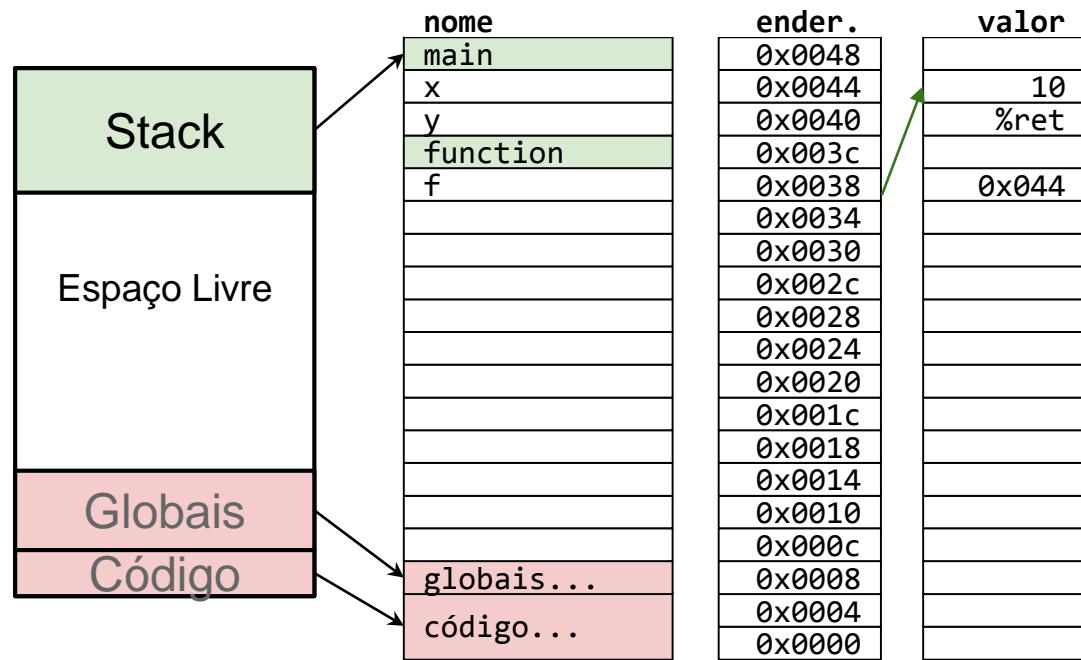


# Stack

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



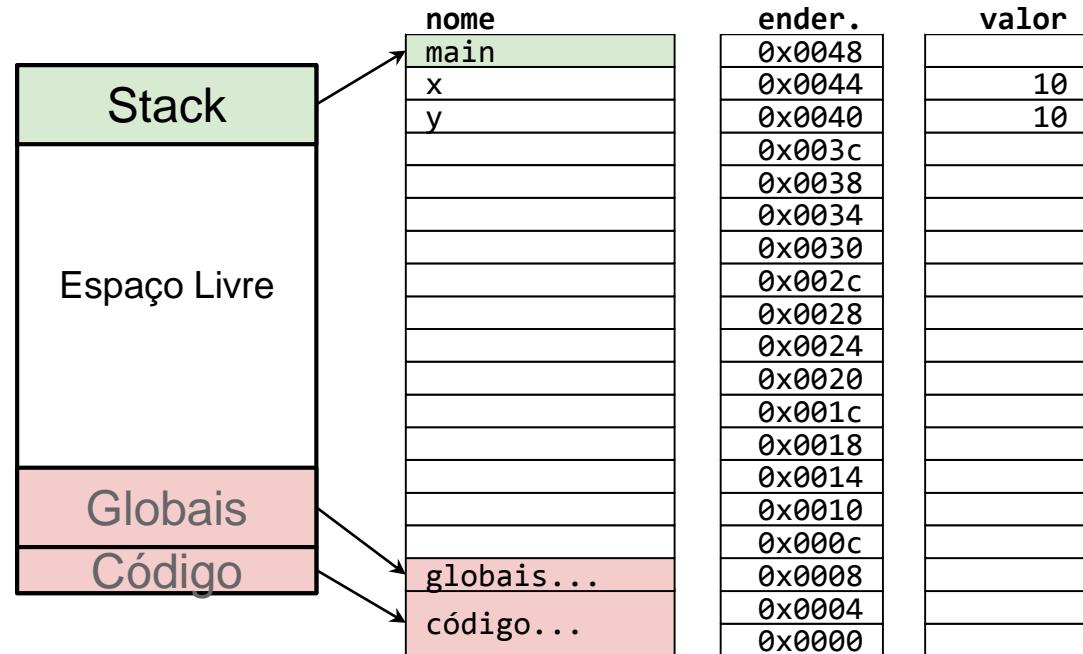
# Stack

## Voltamos para o main

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



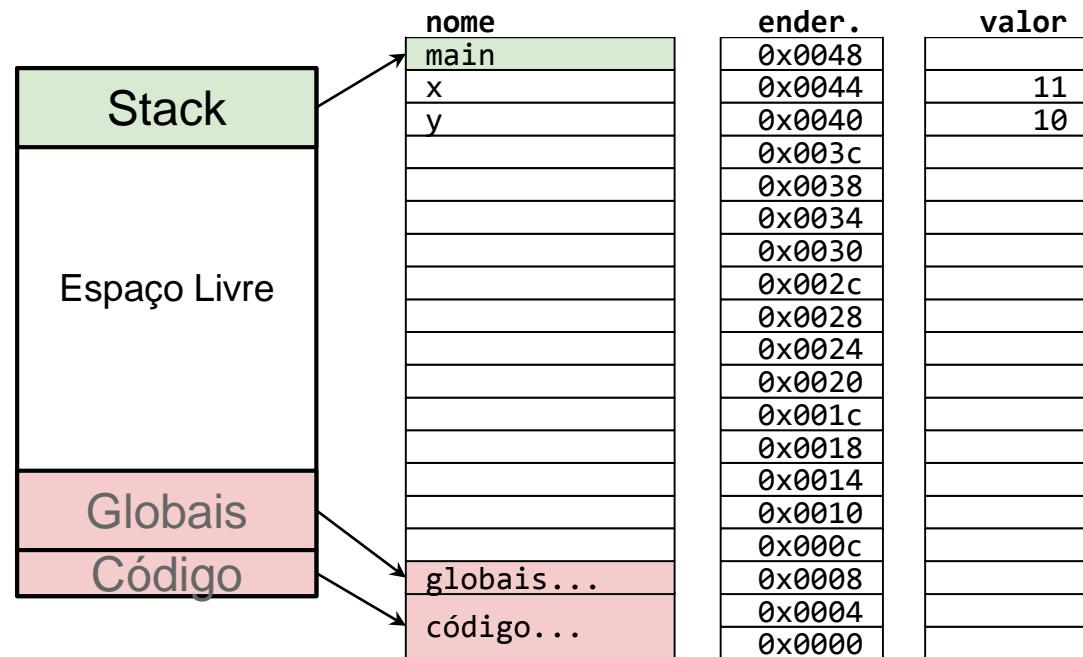
# Stack

x e y estão em posições diferentes da memória

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



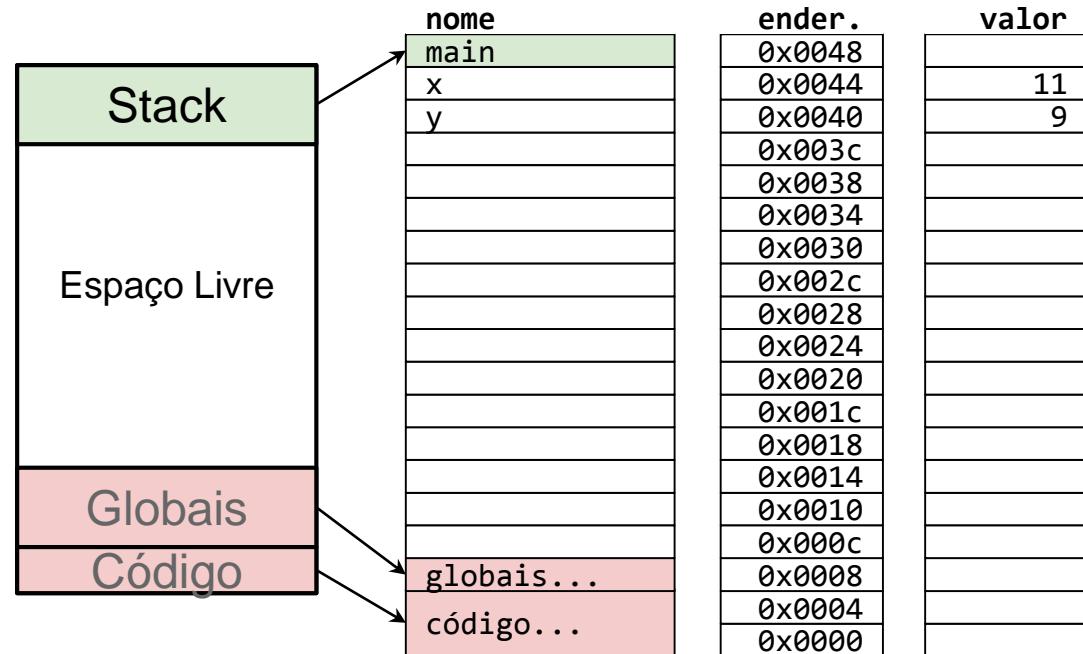
# Stack

x e y estão em posições diferentes da memória

```
#include <iostream>

int function(int *f) {
    *f = *f + 3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```



# Stack/Pilha

Controla o fluxo de execução do programa

- Comportamento similar ao TAD Pilha
  - Para aqueles que viram ED
- Vai guardando a posição das funções, retornos e variáveis

# Referências em C++

C++ Permite o uso de referências

```
#include <iostream>

int function(int &f) {
    f=f+3;
    return f;
}

int main() {
    int x = 7;
    int y;
    y = function(x);
    x++;
    y--;
    return 0;
}
```

Mesmo  
comportamento

```
#include <iostream>

int function(int *f) {
    *f=*f+3;
    return *f;
}

int main() {
    int x = 7;
    int y;
    y = function(&x);
    x++;
    y--;
    return 0;
}
```

# Referências & e ponteiros \*

Quase iguais...

- Assim como em C: & (address of)
- ❑ Porém, podemos usar como:
  - ❑ Ponteiro
    - ❑ `int *valor_ptr = &valor;`
  - ❑ Referência
    - ❑ `int &valor_ptr = &valor;`

# Referências & e ponteiros \*

Quando usar cada um

- Ao alocar memória no heap,  
é comum utilizar \*
  - malloc em C
  - new em C++
- Referências (&) representa um ponteiro  
imutável útil para evitar bugs
  - Boa prática em funções

# Exemplos

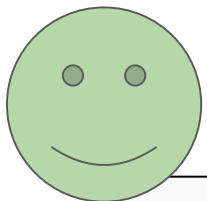
## Duas formas de passar por referência em C++

```
void function(int &i) {  
    int j = 10;  
    i = &j;  
}
```

```
void function(int *i) {  
    int j = 10;  
    i = &j;  
}
```

# Exemplos

## Duas formas de passar por referência em C++



Erro de compilação

```
void function(int &i) {  
    int j = 10;  
    i = &j;  
}
```



Compila só que é uma ótima fonte de bugs!

```
void function(int *i) {  
    int j = 10;  
    i = &j;  
}
```

# Exemplo (por valor)

```
#include <iostream>

int inc(int x) {
    return ++x;
}

int main() {
    int a = 10;
    int b = inc(a);
    std::cout << a << std::endl;
    std::cout << b << std::endl;
    std::cout << &a << std::endl;
    std::cout << &b << std::endl;
}
```

# Exemplo (por referência estilo C)

```
#include <iostream>

int inc(int *x) {
    *x = *x + 1;
    return *x;
}

int main() {
    int a = 10;
    int b = inc(&a);
    std::cout << a << std::endl;
    std::cout << b << std::endl;
    std::cout << &a << std::endl;
    std::cout << &b << std::endl;
}
```

# Exemplo (por referência estilo C++)

```
#include <iostream>

int inc(int &x) {
    x++;
    return x;
}

int main() {
    int a = 10;
    int b = inc(a);
    std::cout << a << std::endl;
    std::cout << b << std::endl;
    std::cout << &a << std::endl;
    std::cout << &b << std::endl;
}
```

# Heap

Memória alocada dinamicamente

- Uma forma simples de pensar:
  - Memória da pilha é gerida automaticamente
  - Só que temos menos controle
  - Nem sempre podemos usar a pilha
- O Heap é onde mora a memória dinâmica
  - Em C utilizamos malloc
  - Em C++ utilizamos new

# Heap

## Liberação de memória

- O programador que libera a memória
  - Em C utilizamos free
  - Em C++ utilizamos delete

# Exemplo

```
#include <cstdlib>
#include <iostream>

int main() {
    int *ptr_a = nullptr;
    ptr_a = new int;
    if (ptr_a == nullptr) {
        std::cout << "Memoria insuficiente!" << std::endl;
        exit(1);
    }

    std::cout << "Endereco de ptr_a: " << ptr_a << std::endl;
    *ptr_a = 90;
    std::cout << "Conteudo de ptr_a: " << *ptr_a << std::endl;
    delete ptr_a;
}
```

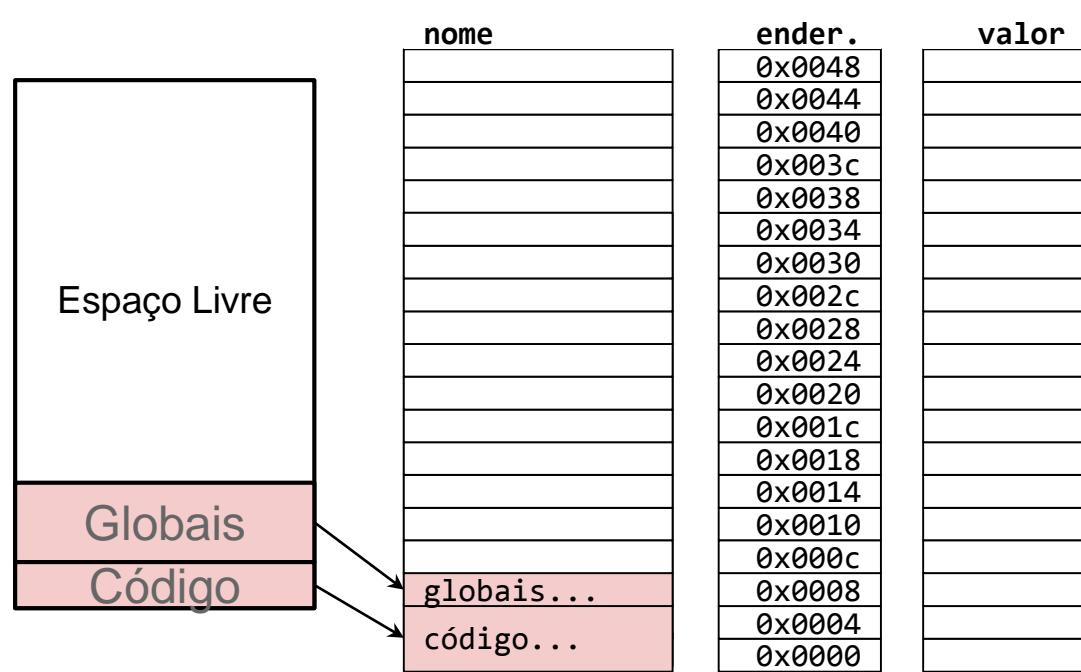
# NULL vs nullptr

- Semanticamente igual
- nullptr é mais seguro
  - Só pode ser atribuído para ponteiros

```
int i = NULL;           // OK, qual o valor de i?  
int i = nullptr;        // Erro de compilação  
int *p = NULL;          // OK, seguro  
int *p = nullptr;        // OK, seguro
```

# Exemplo

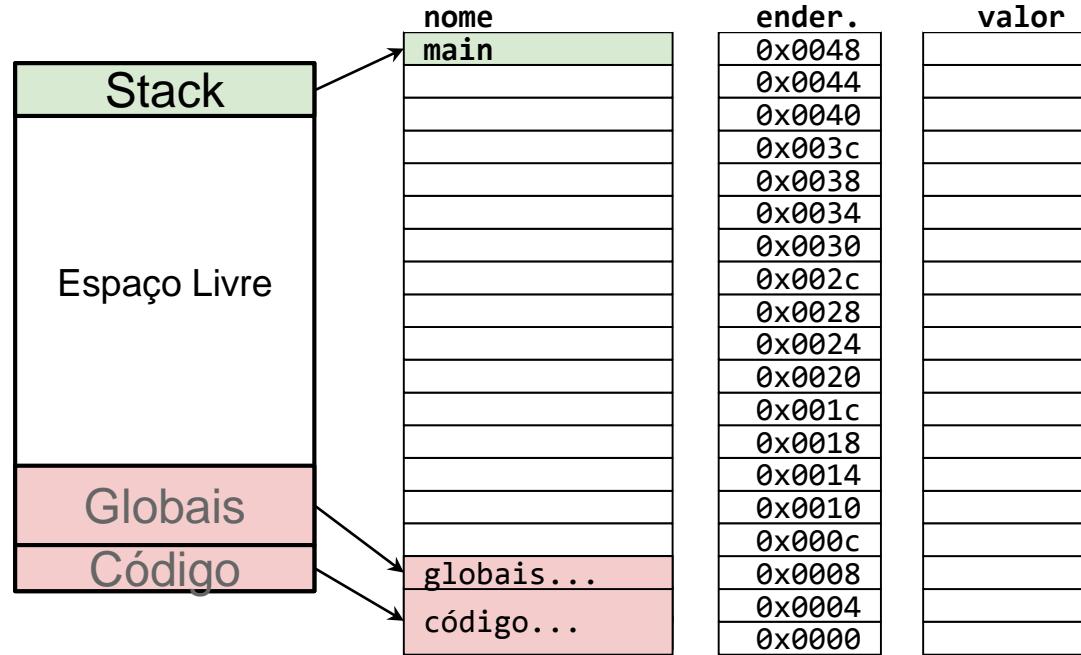
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo



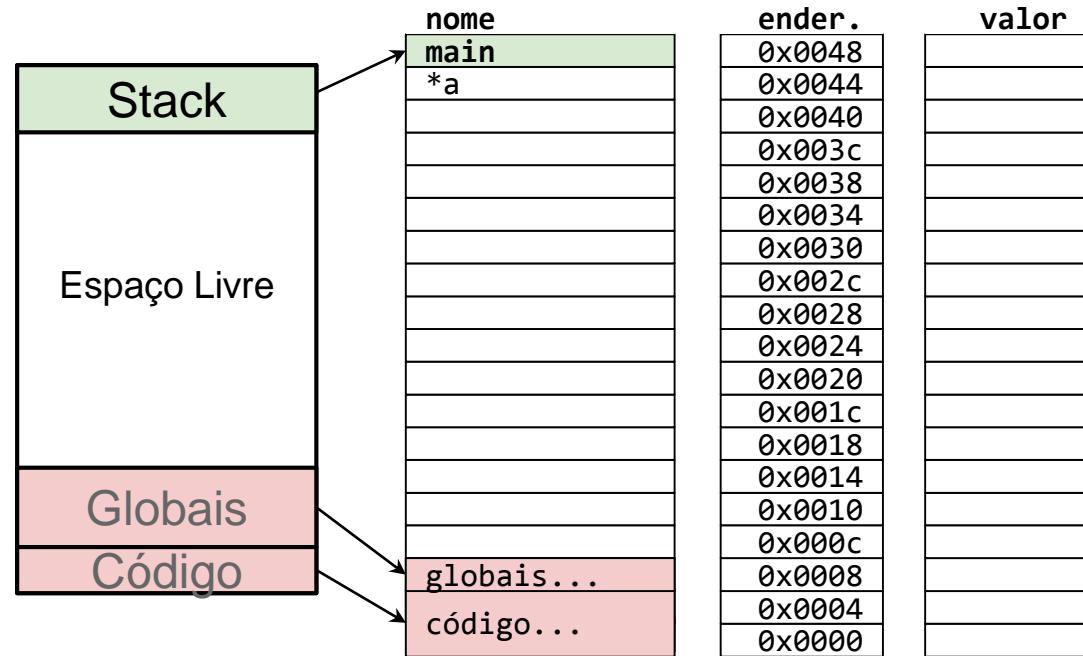
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Alocamos um inteiro no heap, a referência mora a pilha

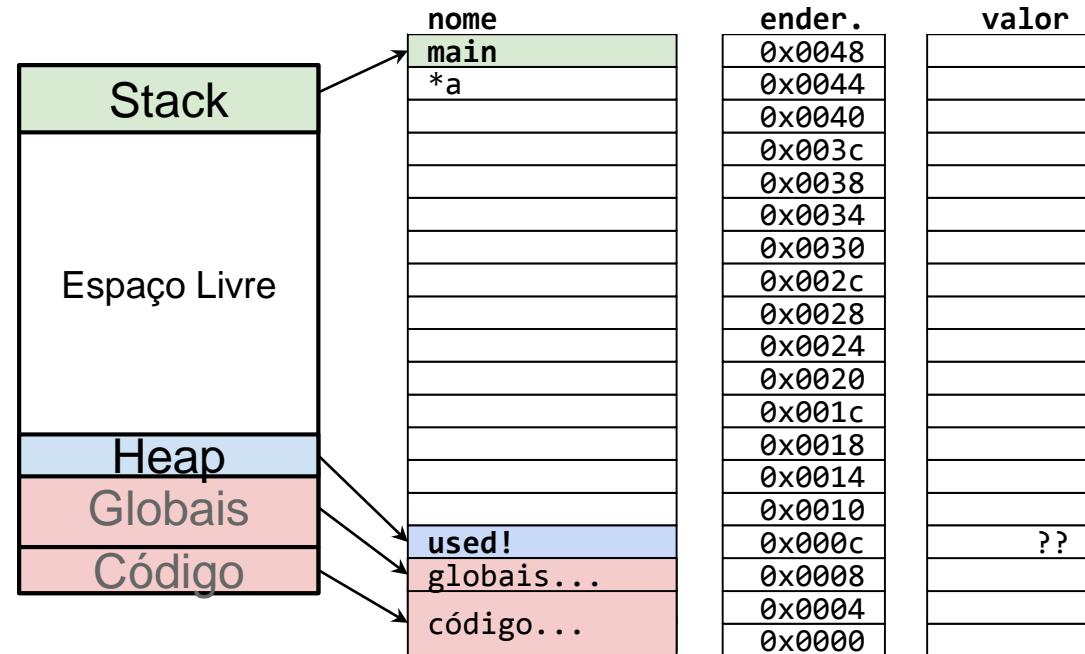
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Alocamos um inteiro no heap, a referência mora a pilha

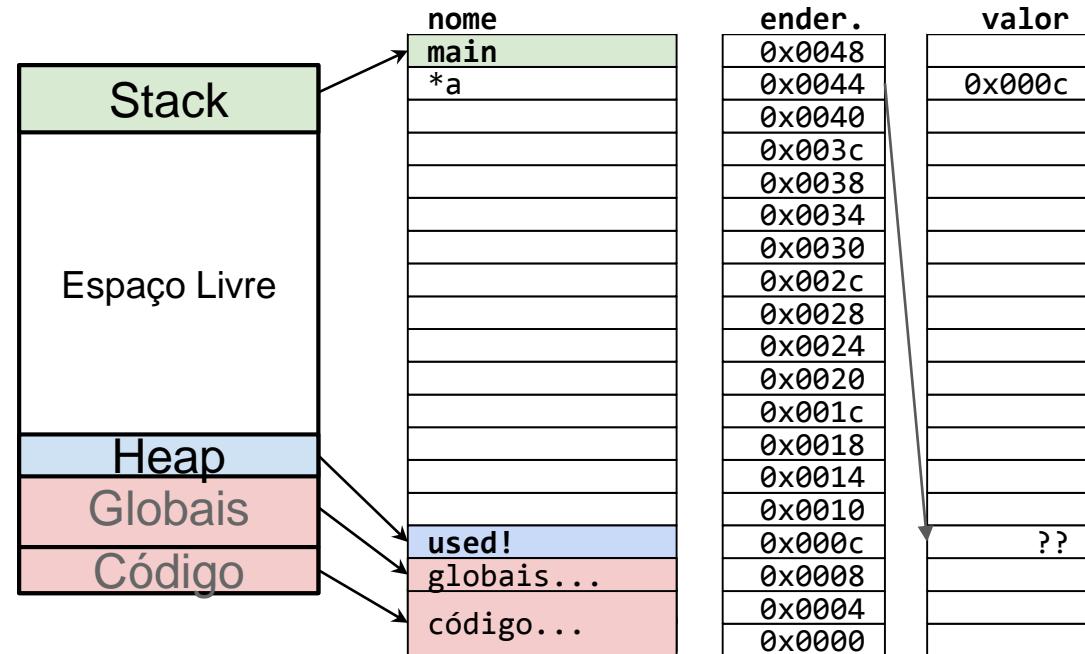
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Alocamos um inteiro no heap, a referência mora a pilha

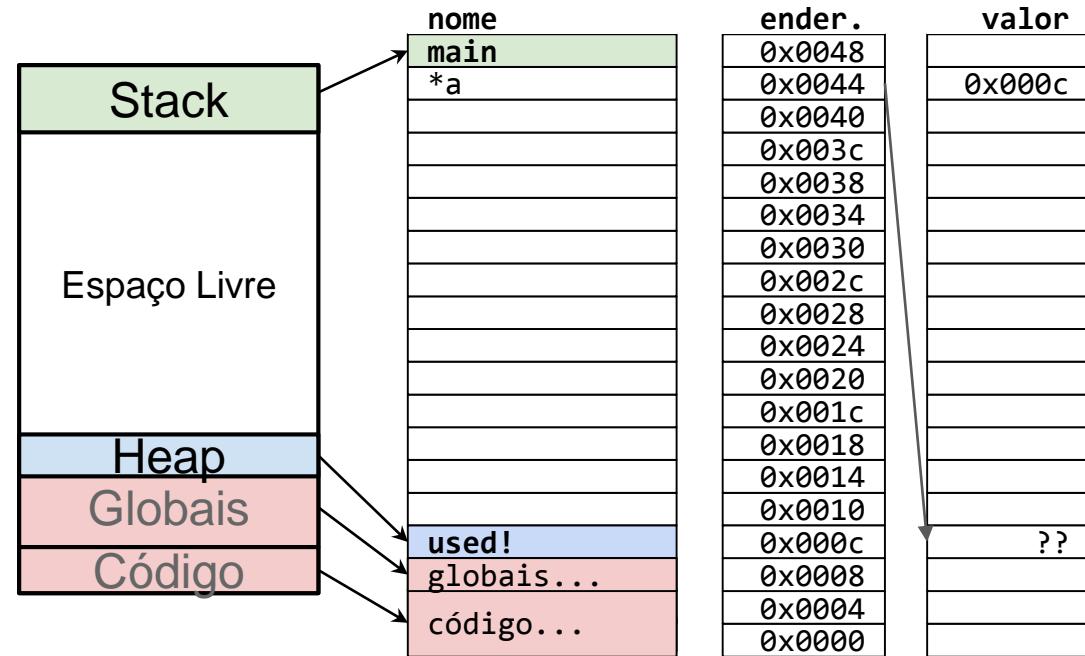
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Não temos variável no heap! Acesso por \*a

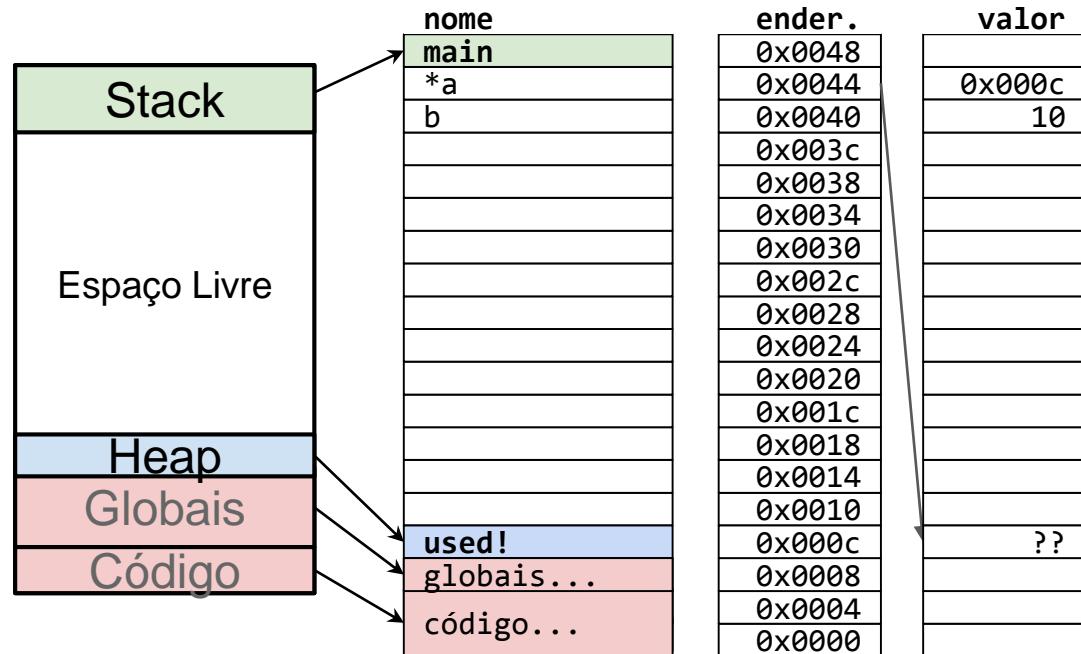
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

b mora na pilha

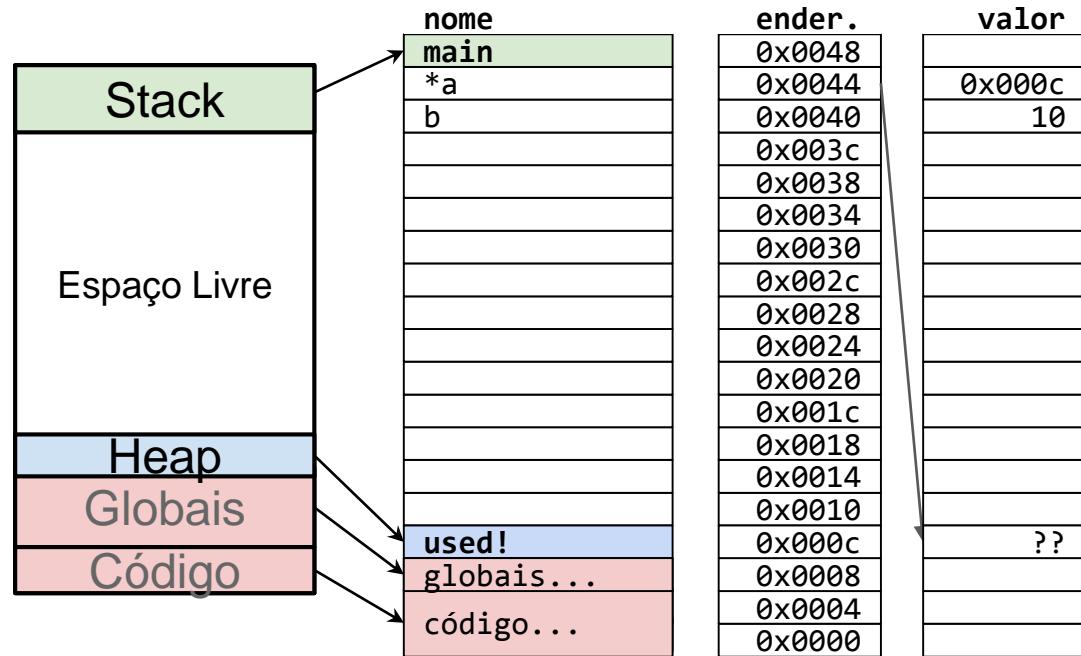
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Como que o diagrama muda?

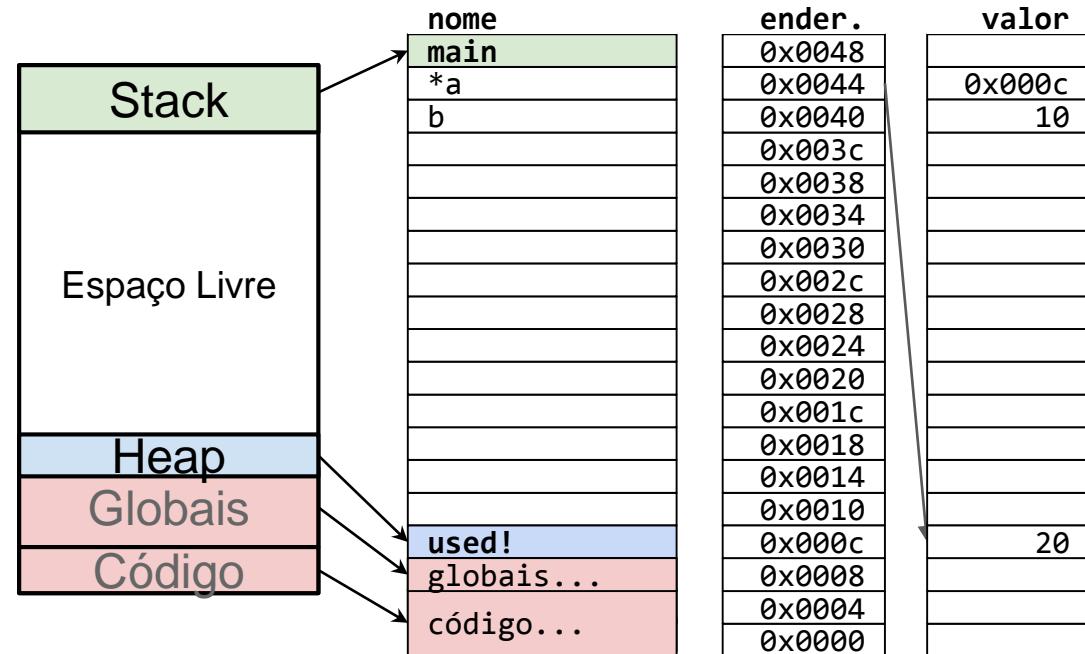
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Valor do local de memória referenciado por a = 20

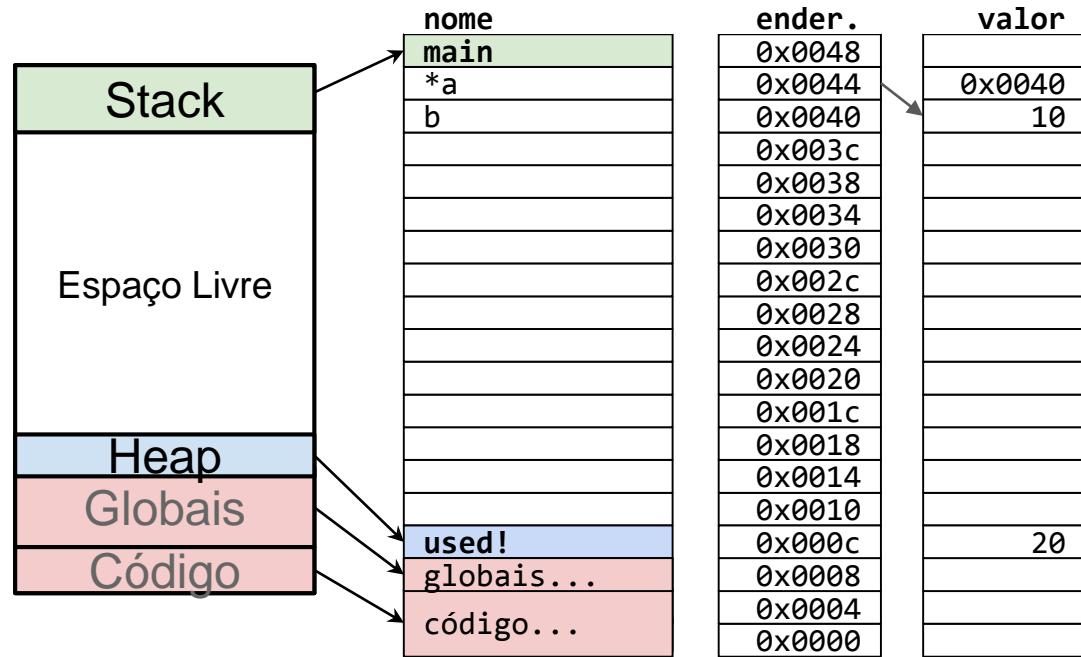
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Como que o diagrama muda?

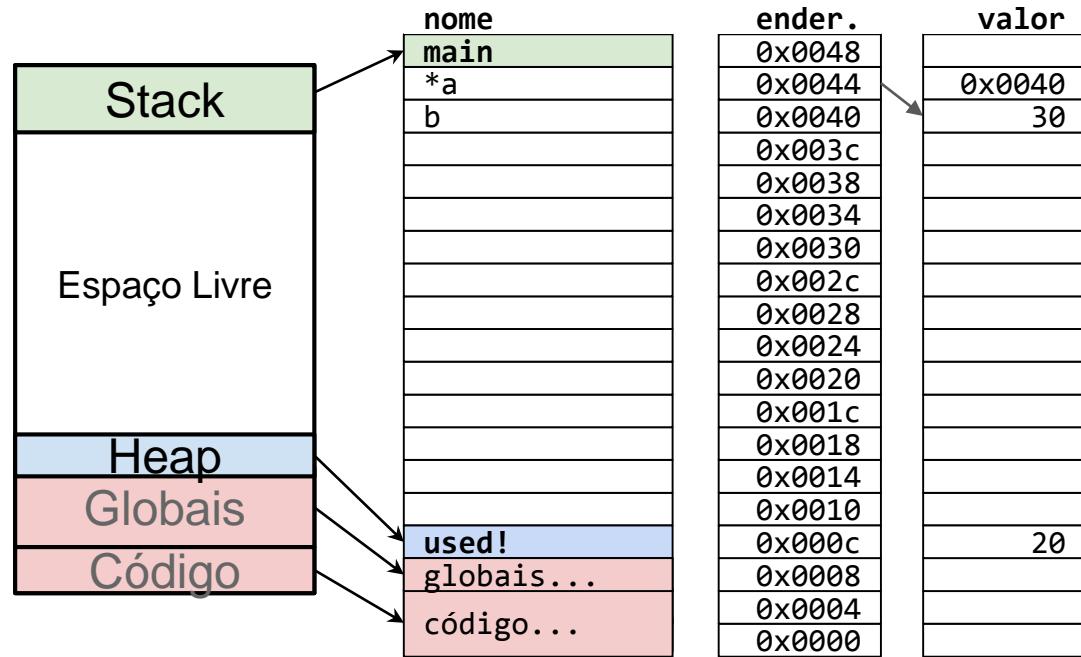
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    → a = &b;  
    *a = 30;  
}
```



# Exemplo

Como que o diagrama muda?

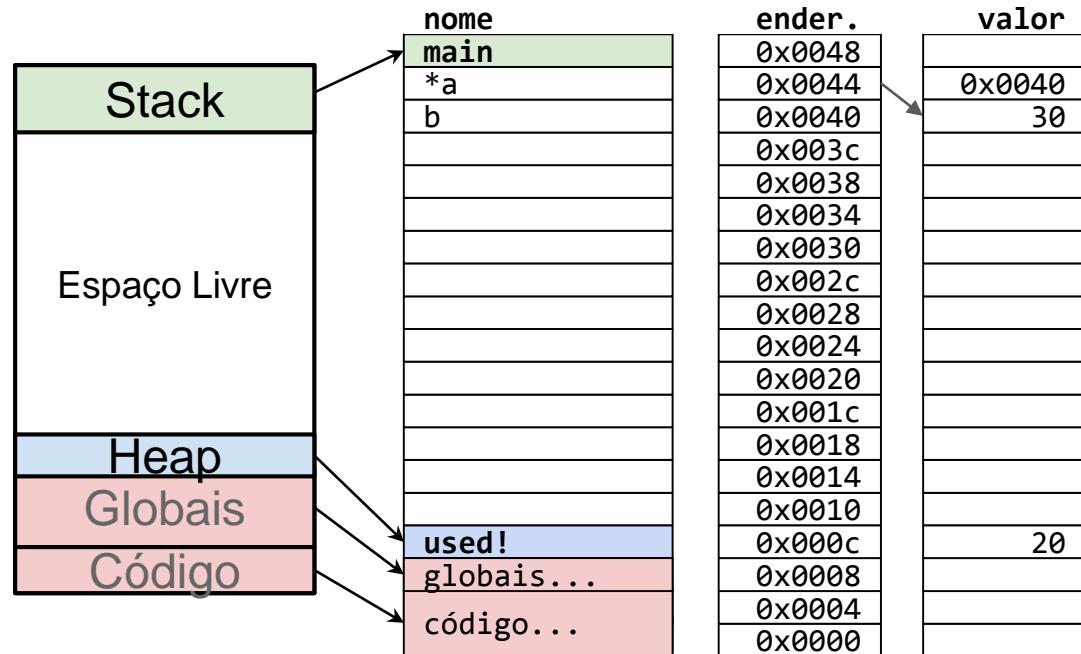
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Qual o problema?

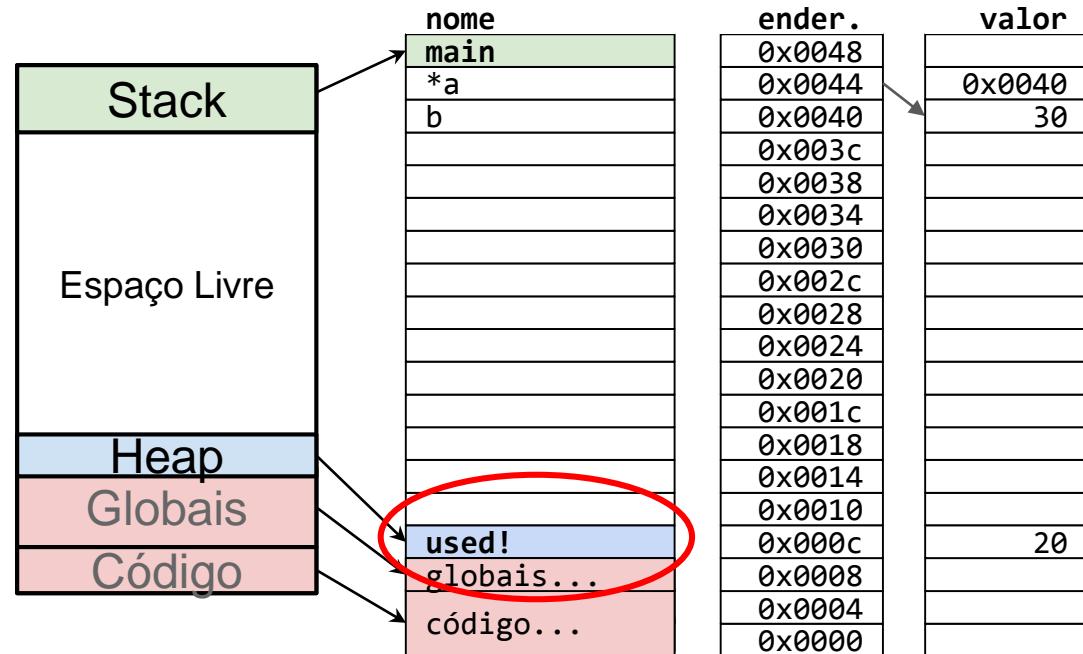
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Qual o problema?

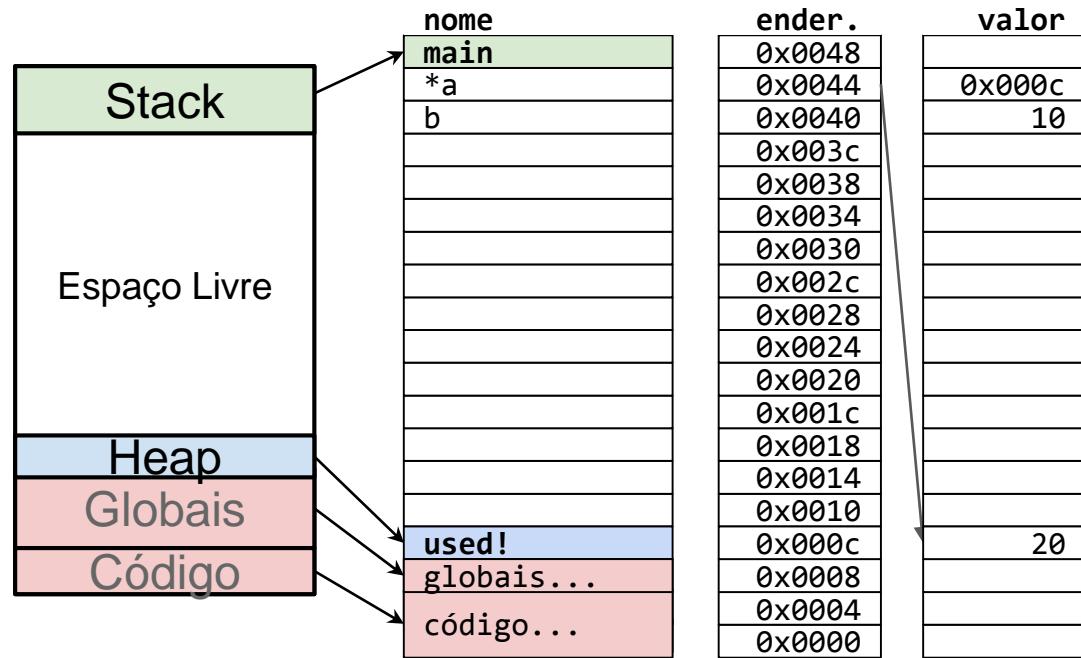
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Qual o problema?

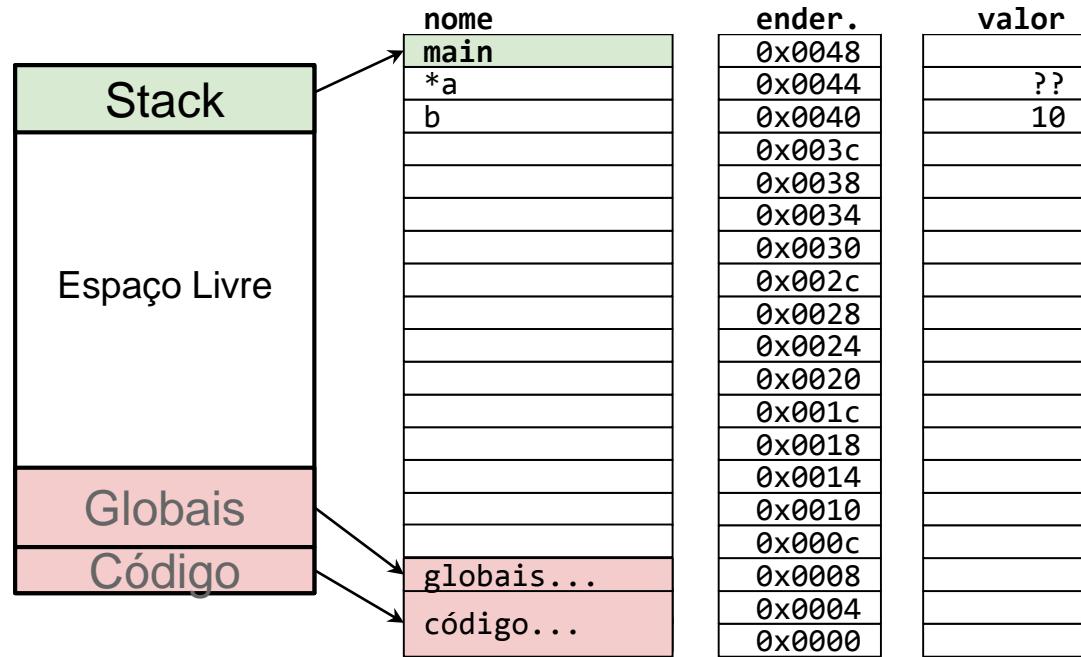
```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    delete a;  
    a = &b;  
    *a = 30;  
}
```



# Exemplo

Qual o problema?

```
int main() {  
    int *a = new int;  
    int b = 10;  
    *a = 20;  
    delete a;  
    a = &b;  
    *a = 30;  
}
```



# Alocação dinâmica de vetores

- Normalmente, a alocação dinâmica é utilizada para criar vetores em tempo de execução
- Exemplo:

```
int *p = new int[10];
```

- Aloca um vetor de inteiros com 10 posições. A manipulação é feita normalmente:  $p[i] = \dots$
- O apontador  $p$  guarda o endereço (aponta) da primeira posição do vetor.

# Exemplo vetores

```
#include <cstdlib>
#include <iostream>

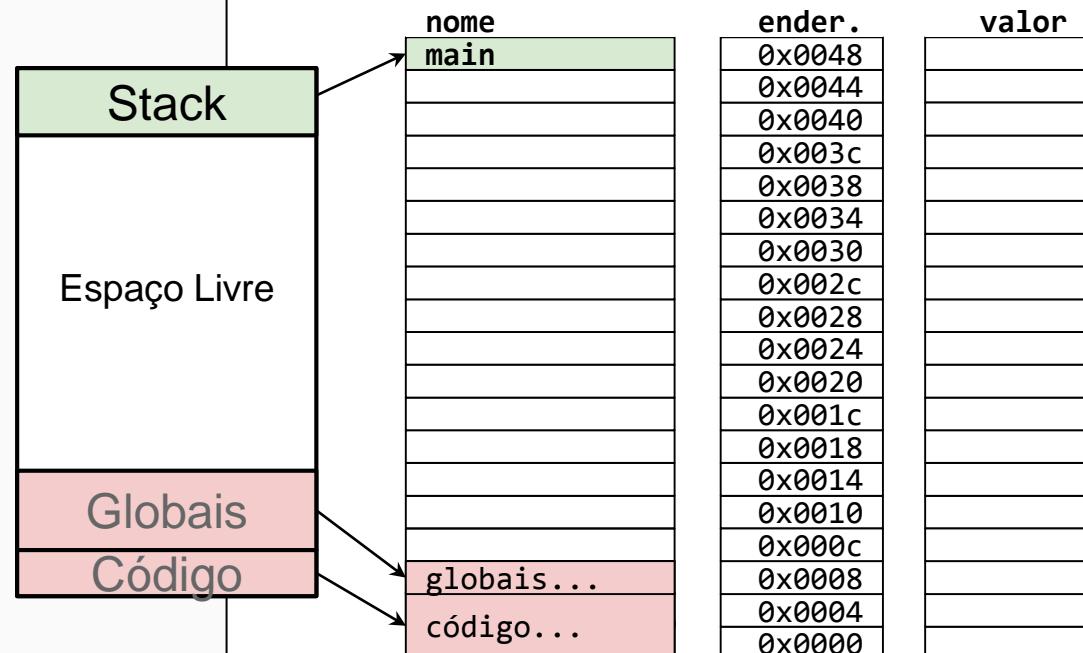
int *produto_interno(int n, int *vetor_a, int *vetor_b) {
    int *resultado = new int[n];
    for (int i = 0; i < n; i++)
        resultado[i] = vetor_a[i] * vetor_b[i];
    return resultado;
}

int main() {
    int vetor_a[3] = {1, 2, 3};
    int vetor_b[3] = {3, 2, 1};
    int *resultado = produto_interno(3, vetor_a, vetor_b);
    for (int i = 0; i < 3; i++)
        std::cout << resultado[i];
    delete[] resultado; // Para vetores usamos delete[]
}
```

# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

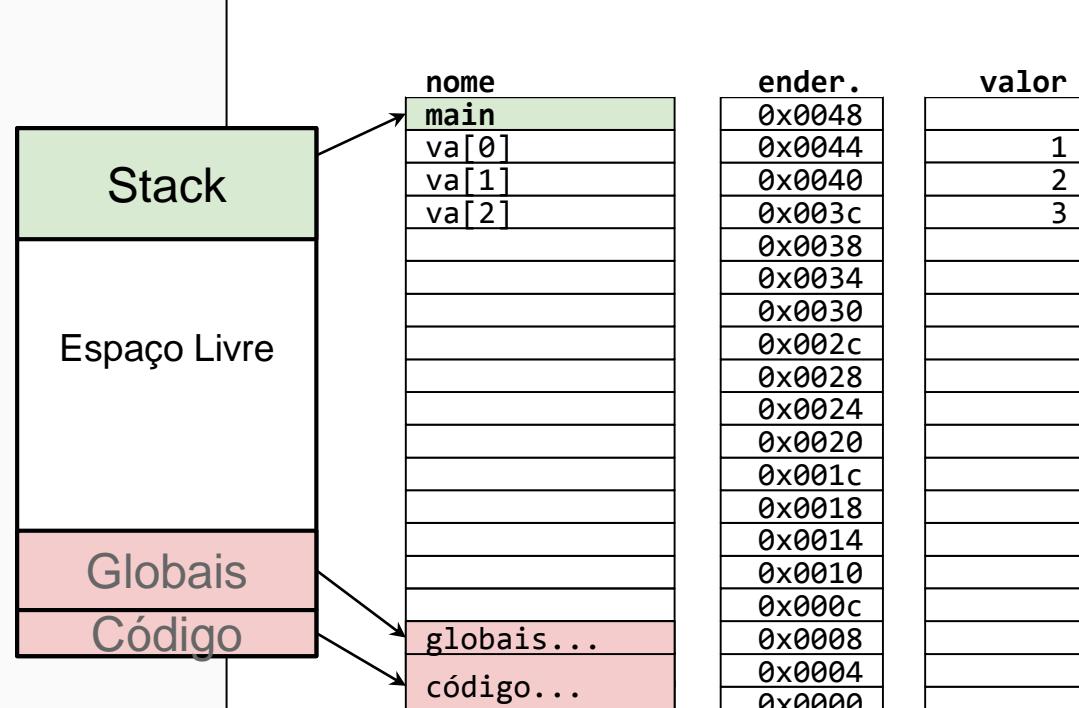
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

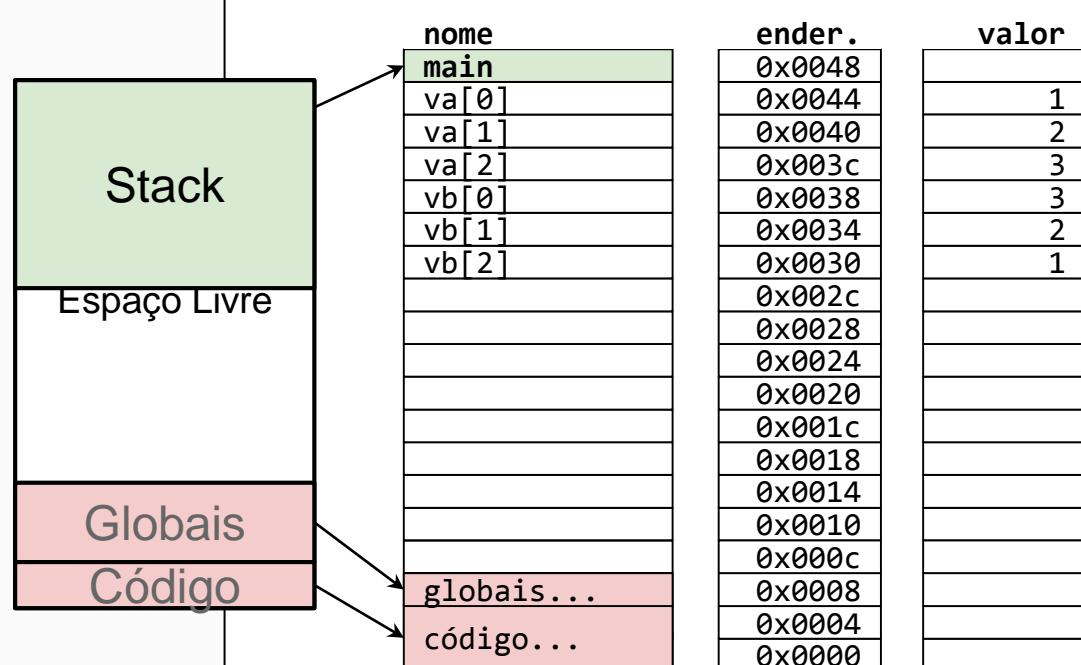
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

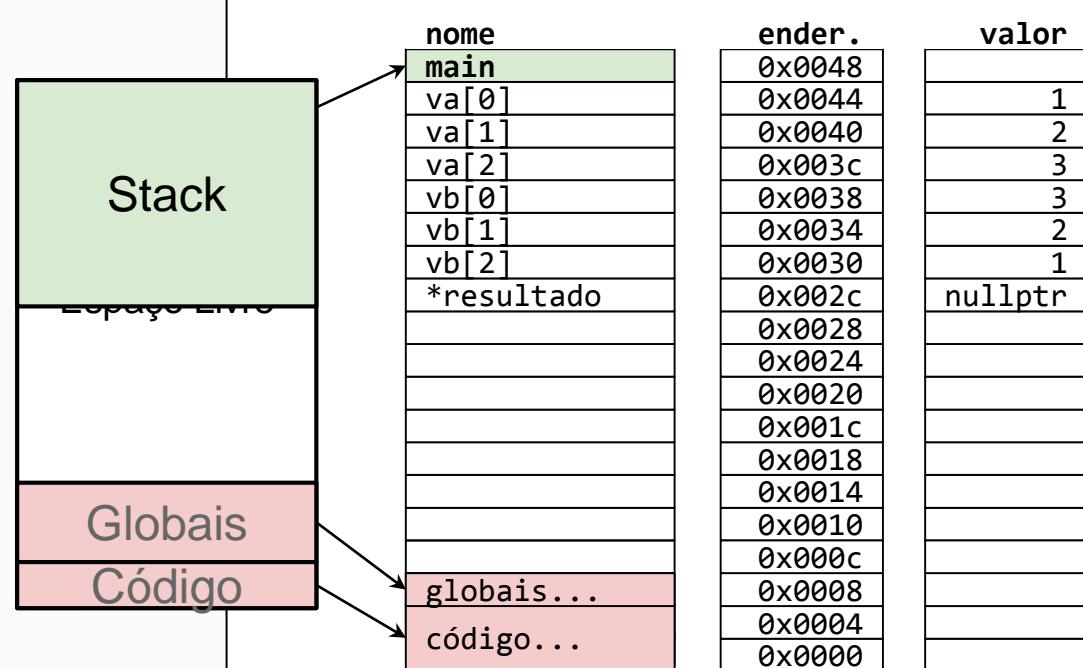
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

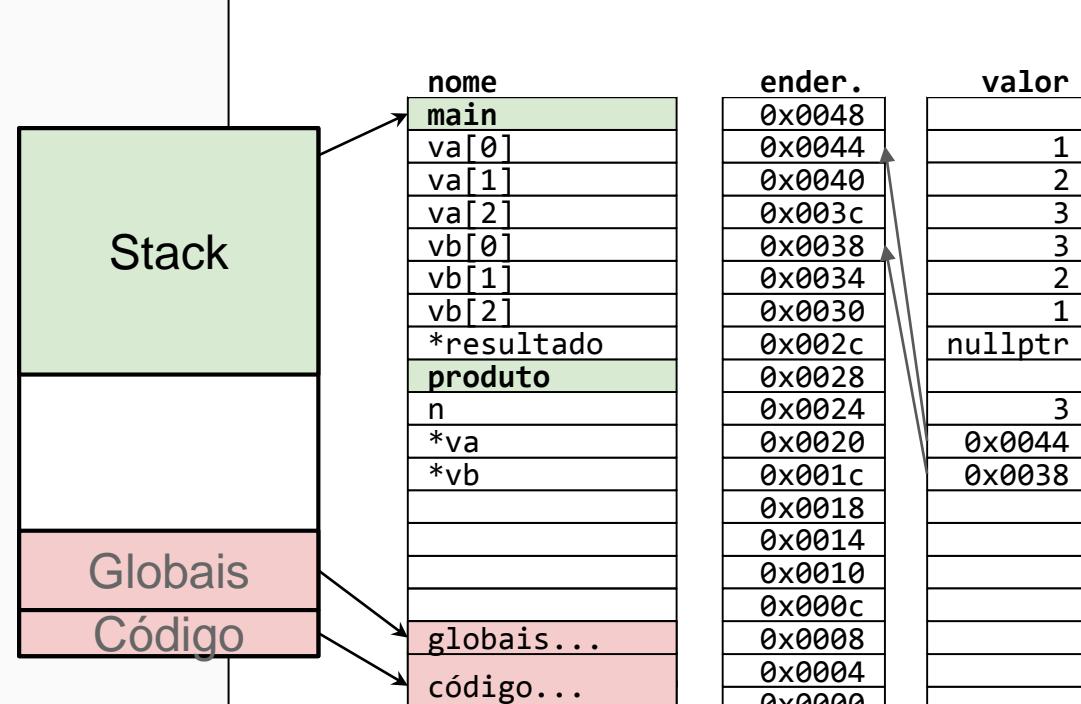
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

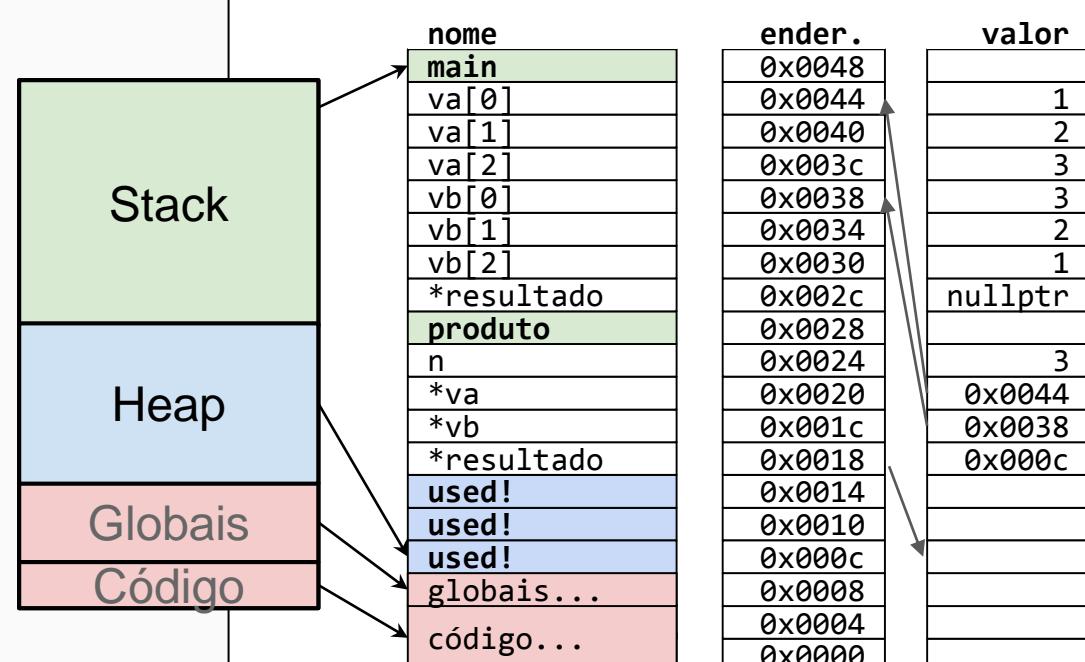
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

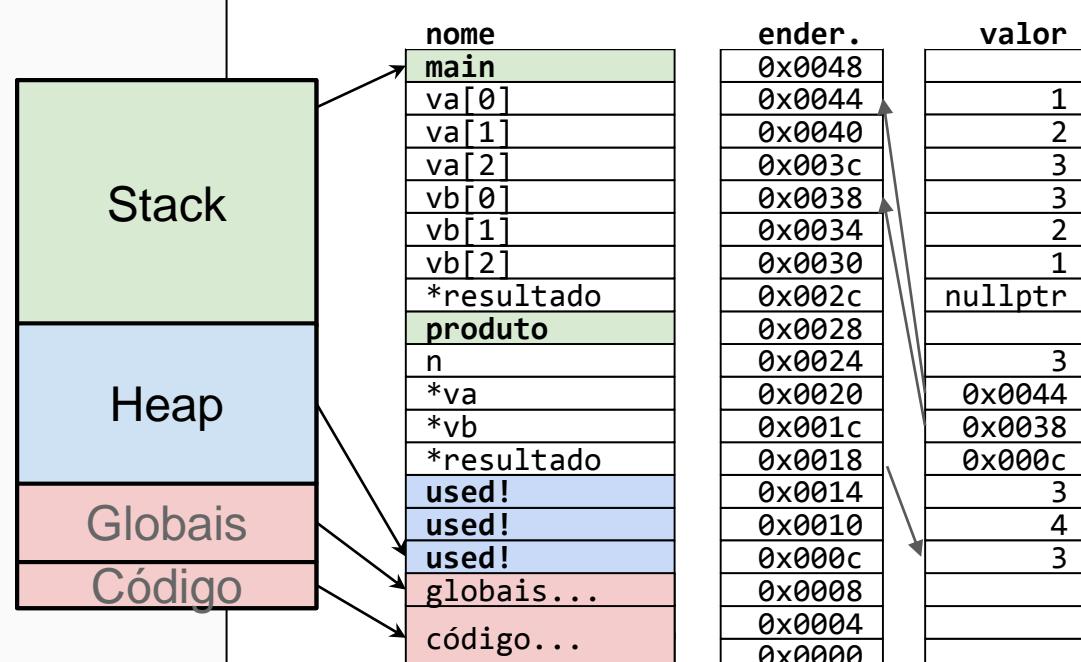
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

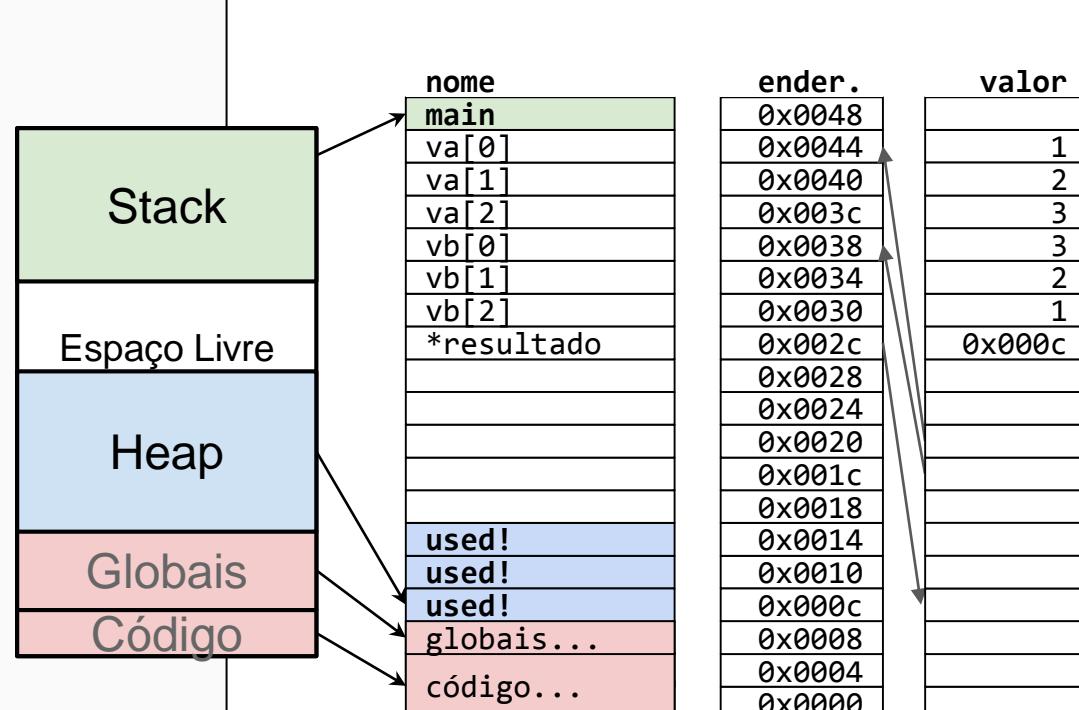
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

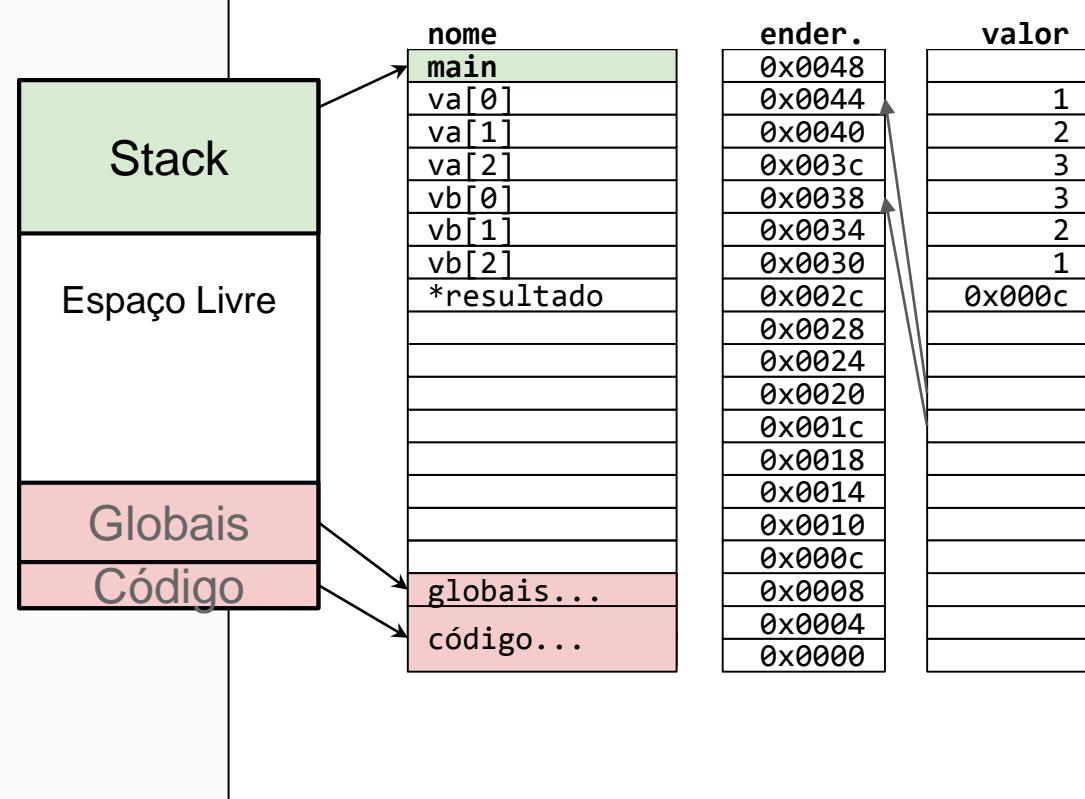
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



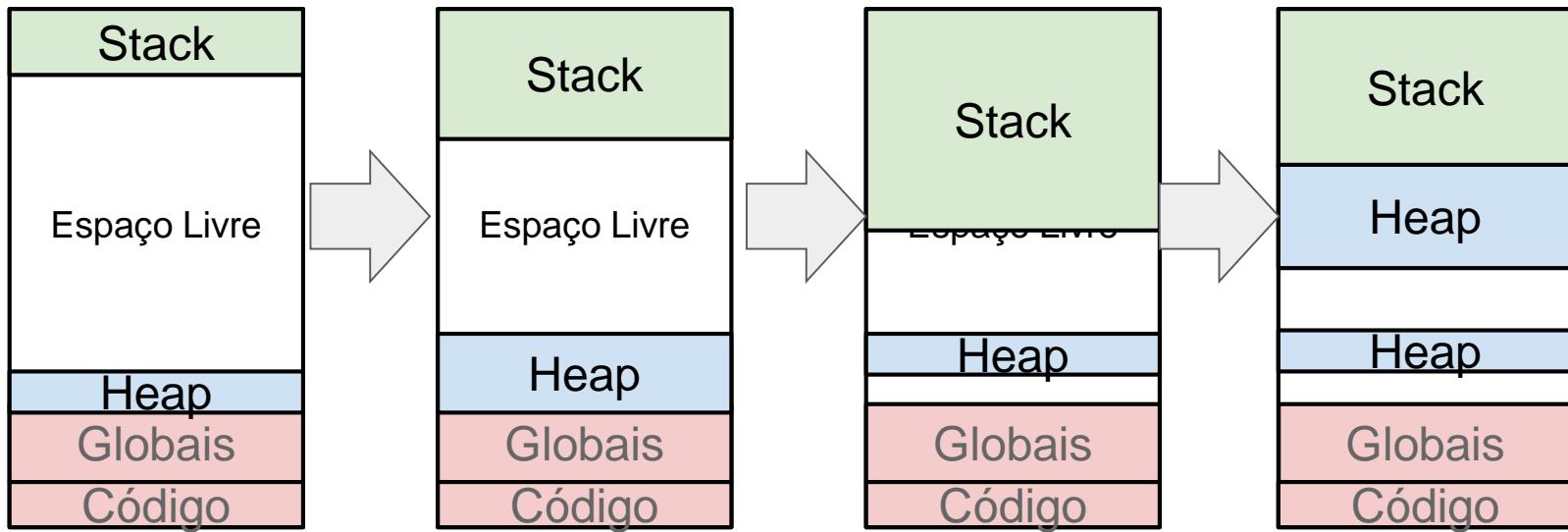
# Exemplo vetores

```
int *produto(int n, int *va, int *vb) {  
    int *resultado = new int[n];  
    for (int i = 0; i < n; i++)  
        resultado[i] = va[i] * vb[i];  
    return resultado;  
}
```

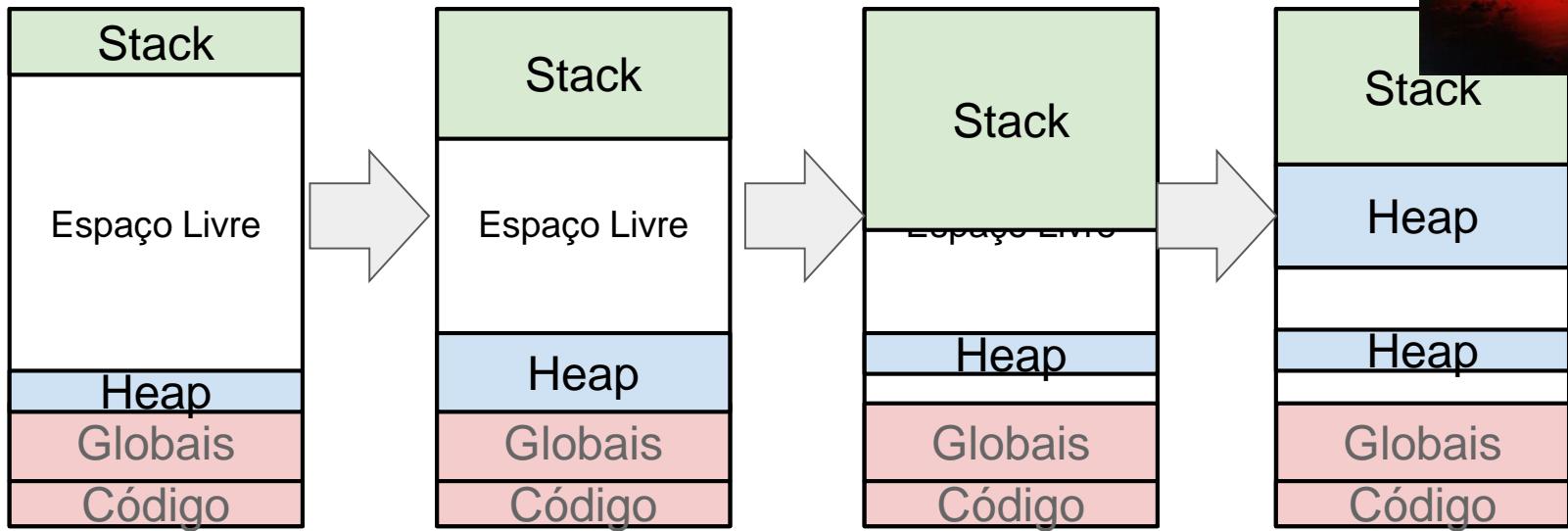
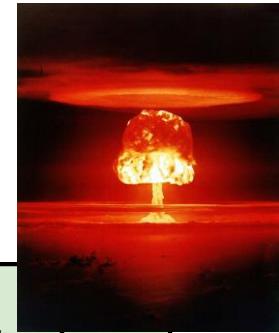
```
int main() {  
    int va[3] = {1, 2, 3};  
    int vb[3] = {3, 2, 1};  
    int *resultado = nullptr;  
    resultado = produto(3, va, vb);  
    for (int i = 0; i < 3; i++)  
        std::cout << resultado[i];  
    delete[] resultado;  
}
```



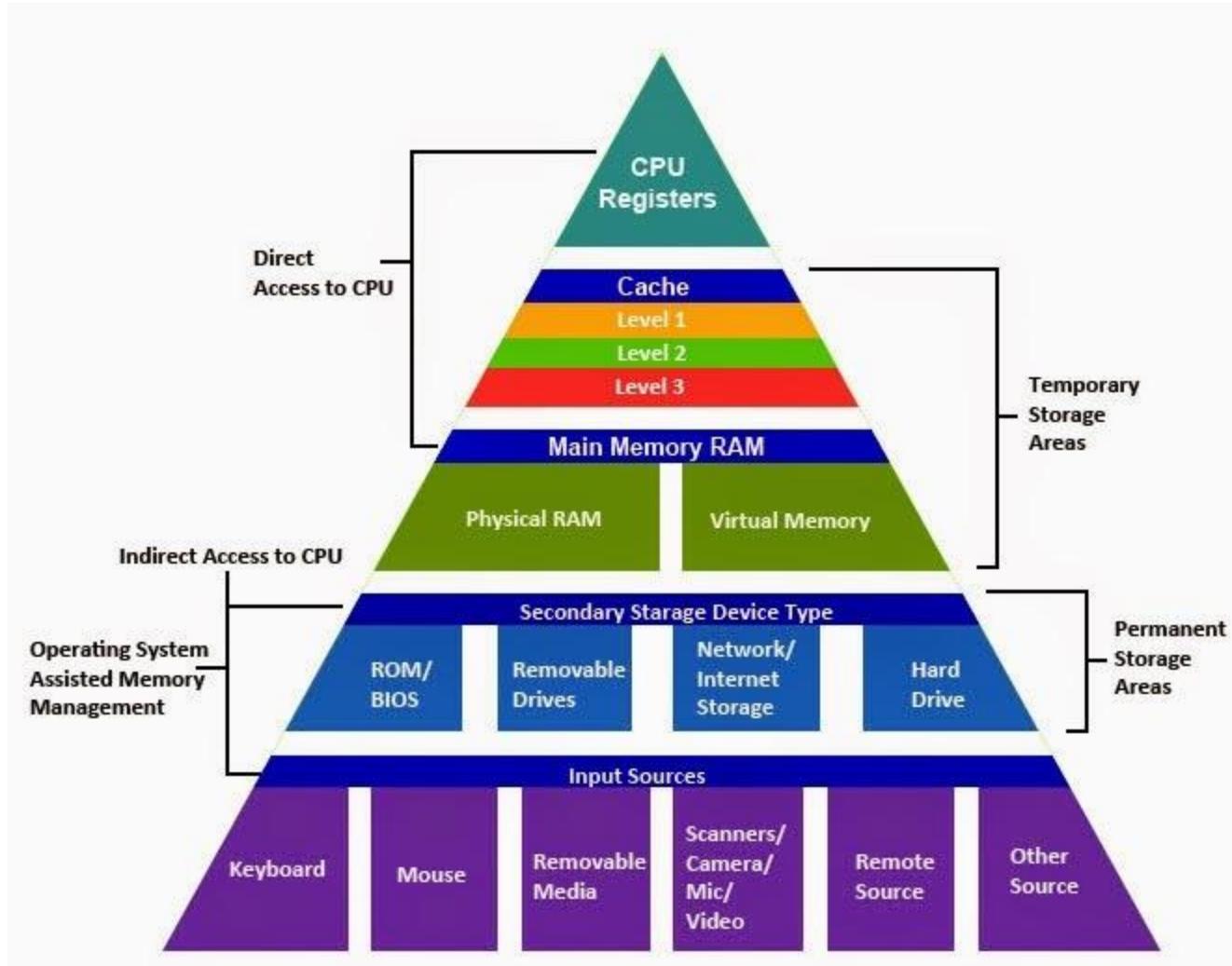
# Sempre libere o heap



# Sempre libere o heap

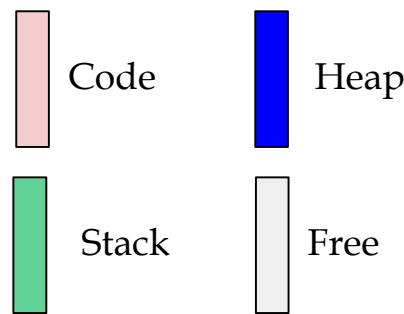


# Hierarquia de Memória



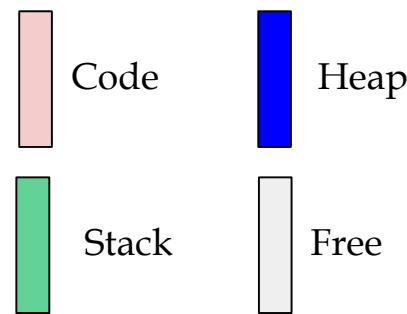
# Memória virtual

Memória na prática!

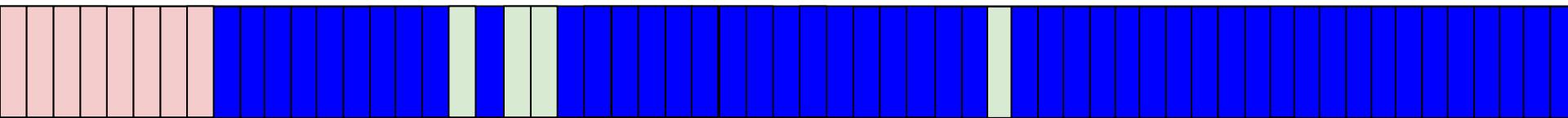


- Embora pareça contígua para o programa
- A memória é na verdade toda fatiada
- O conceito de “endereços virtuais”
  - Faz com que tudo pareça sequencial (contígua)

# Memória virtual



Memória na prática!



- Em situações limite vamos ter que usar o disco da máquina, lento
- Portanto a gerência de memória é importante

# Erros comuns

- Esquecer de alocar memória e tentar acessar o conteúdo da variável
- Copiar o valor do apontador ao invés do valor da variável apontada
- Esquecer de desalocar memória
  - Ela é desalocada ao fim do programa ou procedimento função onde a variável está declarada, mas pode ser um problema em loops
- Tentar acessar o conteúdo da variável depois de liberar a mesma

# Programação e Desenvolvimento de Software 2

## Tipos Abstratos de Dados

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# **Revisando Structs**

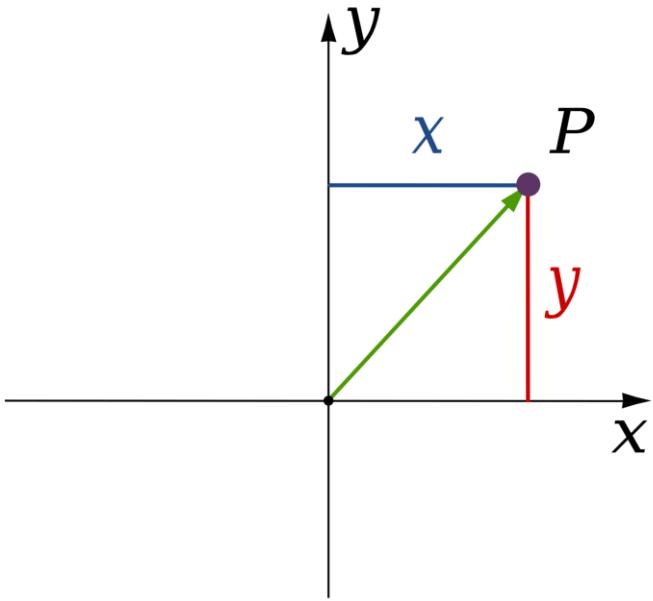
---

# Structs

- Em C/C++ podemos criar novos tipos
- Úteis para representar conceitos mais complexos

# Struct Ponto

apenas x e y



```
struct ponto_t {  
    double x;  
    double y;  
};
```

# structs em C++

- Um pouco mais simples do que em C
- Por enquanto, não precisamos de typedef

```
#include <iostream>
struct ponto_t {
    float x;
    float y;
};

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}
```

# structs em C++

```
#include <iostream>
struct ponto_t {
    float x;
    float y;
};

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}
```

nome	end(&)	val(*)
main	0x0048	
	0x0044	
	0x0040	
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# structs em C++

```
#include <iostream>
struct ponto_t {
    float x;
    float y;
};

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}
```

nome	end(&)	val(*)
main	0x0048	??
ponto_a.x	0x0044	??
ponto_b.y	0x0040	
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# structs em C++

```
#include <iostream>
struct ponto_t {
    float x;
    float y;
};

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}
```

nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	7
ponto_b.y	0x0040	??
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# structs em C++

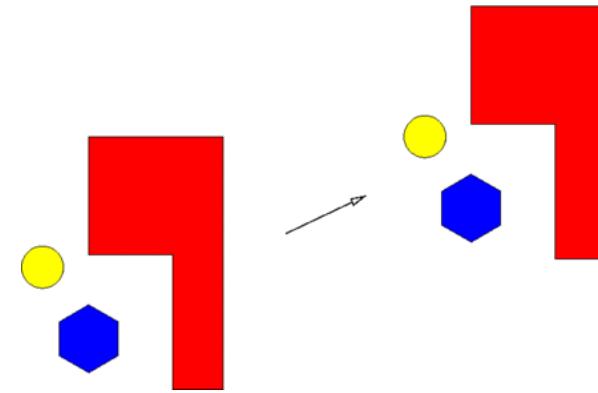
```
#include <iostream>
struct ponto_t {
    float x;
    float y;
};

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}
```

nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	7
ponto_b.y	0x0040	9
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Passagem por referência de structs

- Vamos implementar um procedimento de translação



```
void translacao(ponto_t &ponto, float dx, float dy) {  
    ponto.x += dx;  
    ponto.y += dy;  
}
```

```

#include <iostream>

struct ponto_t {
    float x;
    float y;
};

void translacao(ponto_t &ponto, float dx,
                float dy) {
    ponto.x += dx;
    ponto.y += dy;
}

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    translacao(ponto_a, 3, 1);
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}

```

nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	7
ponto_a.y	0x0040	9
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

```

#include <iostream>

struct ponto_t {
    float x;
    float y;
};

void translacao(ponto_t &ponto, float dx,
                float dy) {
    ponto.x += dx;
    ponto.y += dy;
}

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    translacao(ponto_a, 3, 1);
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}

```



nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	7
ponto_a.y	0x0040	9
transl...	0x003c	
&ponto	0x0038	0x0044
dx	0x0034	3
dy	0x0030	1
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

```

#include <iostream>

struct ponto_t {
    float x;
    float y;
};

void translacao(ponto_t &ponto, float dx,
                float dy) {
    ponto.x += dx;   ← Observe que podemos usar .
    ponto.y += dy;
}

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    translacao(ponto_a, 3, 1);
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}

```

Referência

Sem & Referência

nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	10
ponto_a.y	0x0040	10
transl...	0x003c	
&ponto	0x0038	
dx	0x0034	
dy	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

```

#include <iostream>

struct ponto_t {
    float x;
    float y;
};

void translacao(ponto_t &ponto, float dx,
                float dy) {
    ponto.x += dx;
    ponto.y += dy;
}

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    translacao(ponto_a, 3, 1);
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}

```

nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	10
ponto_a.y	0x0040	10
	0x003c	
	0x0038	
	0x0034	
	0x0030	
	0x002c	
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

## Qual o problema com esta chamada?

```
void translacao(ponto_t ponto, float dx, float dy) {  
    ponto.x += dx;  
    ponto.y += dy;  
}
```

```

#include <iostream>

struct ponto_t {
    float x;
    float y;
};

void translacao(ponto_t ponto, float dx,
                float dy) {
    ponto.x += dx;
    ponto.y += dy;
}

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    translacao(ponto_a, 3, 1);
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}

```

**Passando cópia**

nome	end(&)	val(*)
main	0x0048	
ponto_a.x	0x0044	7
ponto_a.y	0x0040	9
transl...	0x003c	
ponto.x	0x0038	7
ponto.y	0x0034	9
dx	0x0030	3
dy	0x002c	1
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

```

#include <iostream>

struct ponto_t {
    float x;
    float y;
};

void translacao(ponto_t ponto, float dx,
                float dy) {
    ponto.x += dx;
    ponto.y += dy;
}

int main() {
    ponto_t ponto_a;
    ponto_a.x = 7;
    ponto_a.y = 9;
    translacao(ponto_a, 3, 1);
    std::cout << ponto_a.x << std::endl;
    std::cout << ponto_a.y << std::endl;
    return 0;
}

```

*Passando cópia*

nome	end(&)	val(*)
main	0x0048	7
ponto_a.x	0x0044	9
ponto_a.y	0x0040	10
transl...	0x003c	10
ponto.x	0x0038	10
ponto.y	0x0034	10
dx	0x0030	3
dy	0x002c	1
	0x0028	
	0x0024	
	0x0020	
	0x001c	
	0x0018	
	0x0014	
	0x0010	
	0x000c	
	0x0008	
	0x0004	
	0x0000	

# Alocando structs no heap

- Fazemos uso de new e delete
- Similar a uma variável
- Uso comum para implementar listas etc.

```
ponto_t *p = new ponto_t;  
delete p;
```

# **Tipos Abstrato de Dados**

---

# TADs e Operações

- Quais operações uma coleção suporta?
- Pense em um conjunto matemático

# TADs e Operações

- Quais operações uma coleção suporta?
- Pense em um conjunto matemático
  - adicionar (união) elemento
  - remover (complemento) elemento
  - interseção
  - número de elementos
  - domínio
  - . . .

# Tipos Abstratos de Dados (TADs)

- Modelo matemático, acompanhado das operações definidas sobre o modelo.
  - conjunto dos inteiros acompanhado das operações de adição, subtração e multiplicação.
- A implementação do algoritmo em uma linguagem de programação exige a representação do TAD em termos dos tipos de dados e dos operadores suportados.

# Contrato

- TADs são contratos
- Funções que operam em cima da memória

# Encapsulamento

- Conceito importante em TADs
- Usuário:
  - Enxerga a interface
  - Não se preocupa, em primeiro momento,  
como é o TAD por baixo

# **Então TADs são structs?**

---

# TADs vs Structs

- **Não!**
- TADs são um conceito mais geral
  - Existem em qualquer tipo de linguagem
- Em C++
  - Sim, mapeiam bem para structs/classes + .h
  - Ou para interfaces
    - Assuntos futuro
    - Vamos fazer uma abordagem bottom up

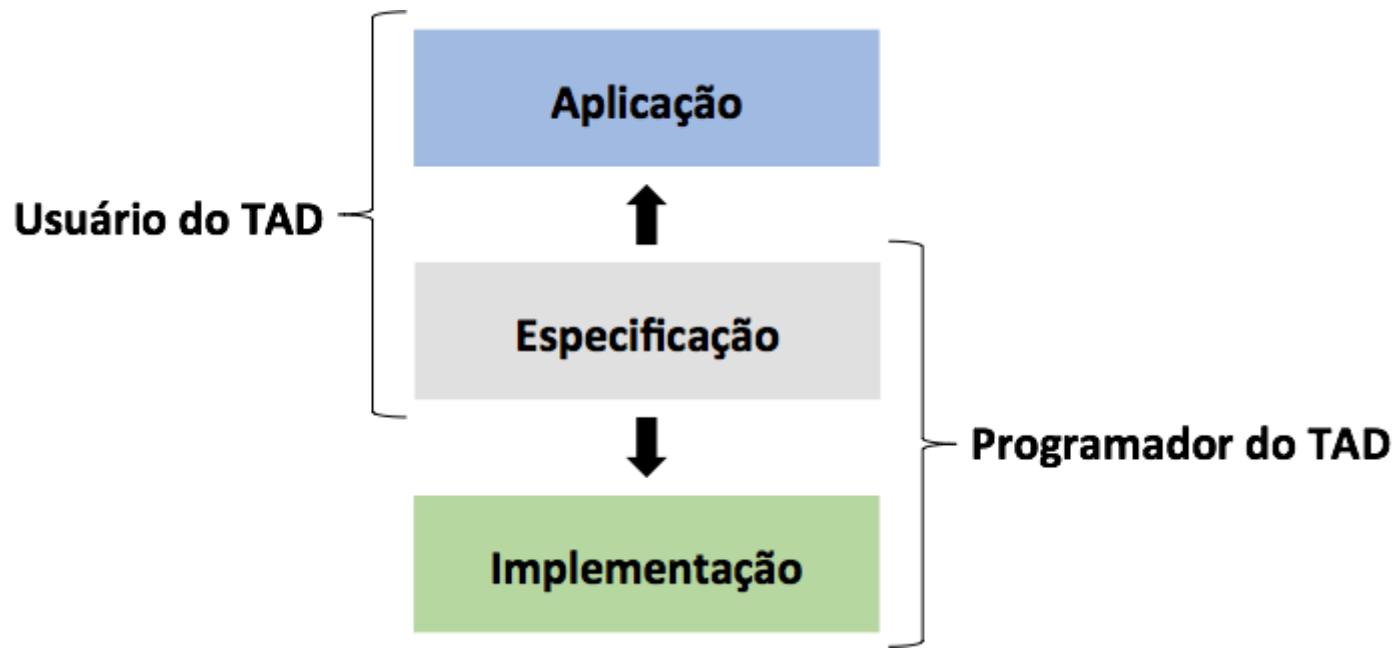
# TADs vs Algoritmos

- Algoritmo
  - Sequência de ações executáveis  
entrada → saída
  - Exemplo: “Receita de Bolo”
  - Algoritmos usam TADs
- TADs
  - Contrato
    - +
  - Memória

# Tipos Abstratos de Dados (TADs)

- Podemos considerar TADs como generalizações de tipos primitivos e procedimentos como generalizações de operações primitivas.
- O TAD encapsula tipos de dados. A definição do tipo e todas as operações ficam localizadas numa seção do programa.
- Os usuários do TAD só tem acesso a algumas operações disponibilizadas sobre esses dados

# Tipos Abstratos de Dados (TADs)



# Tipos Abstratos de Dados (TADs)

- TADs são um conceito de programação
- Vamos aprender como implementar os mesmos usando classes e objetos
- Outras linguagens
  - structs + funções ( C )
  - traits ( Rust )
  - duck typing ( Python, Ruby )
  - classes e interfaces ( Java, C++ )

# Perguntas TAD

Supondo que vamos criar um TAD qualquer

1. Como organizar a memória?

2. **Quais operações?**

a. **Assinaturas**

b. **Contratos**

A primeira pergunta é mais de implementação, o TAD é descrito pela 2.

# **Como fazer um TAD ponto?**

---

# Primeiro problema

- Quais dados temos que representar?

# Primeiro problema

- Quais dados temos que representar?
  - Valor no eixo-x
  - Valor no eixo-y

# Em Código

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
}
```

## Alguns pontos importantes:

1. O uso de `_` é estilo, facilita a codificação
2. Vamos chamar de **Ponto** por estilo também. Isto é, iniciando com maiúsculo

# Até agora temos apenas um struct

## Diferenças de C

1. Não precisamos de **typedef**
2. Podemos associar métodos ao struct

# Segundo Problema

- Quais problemas?
  - Construir o ponto
  - Translação
  - Rotação
  - Imprimir

# Imprimir o ponto

## Operação mais simples

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout << "x= " << _x_val << "y=" << _y_val << std::endl;  
    }  
};
```

# Imprimir o ponto

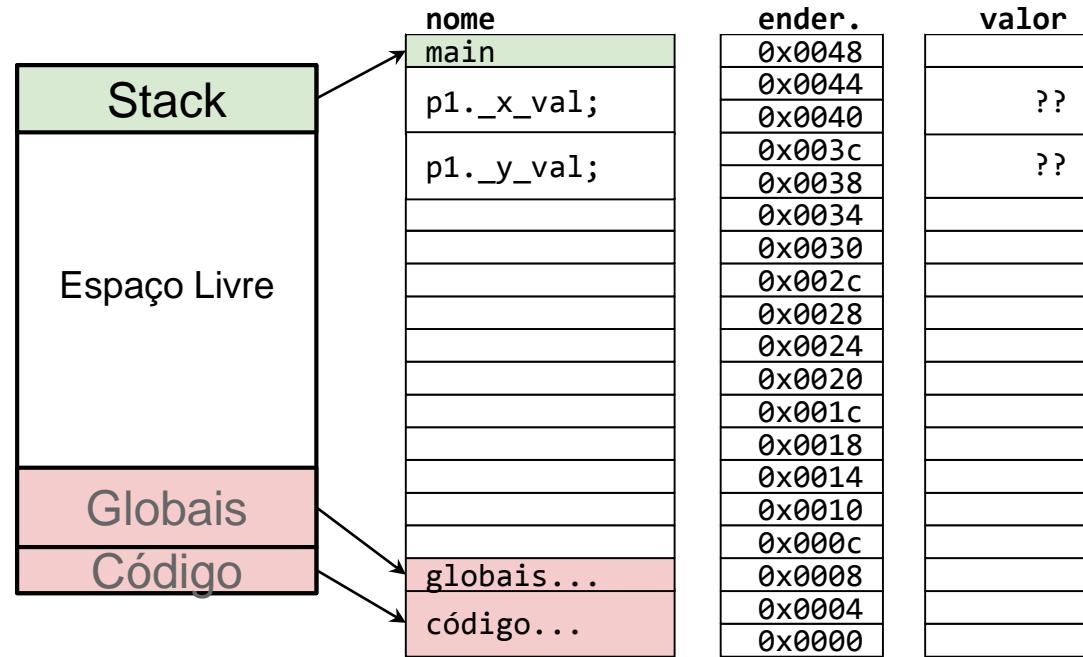
## Operação mais simples (omitindo imports)

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout << "x= " << _x_val << "y=" << _y_val << std::endl;  
    }  
};
```

Note que não recebe nada

# Na memória

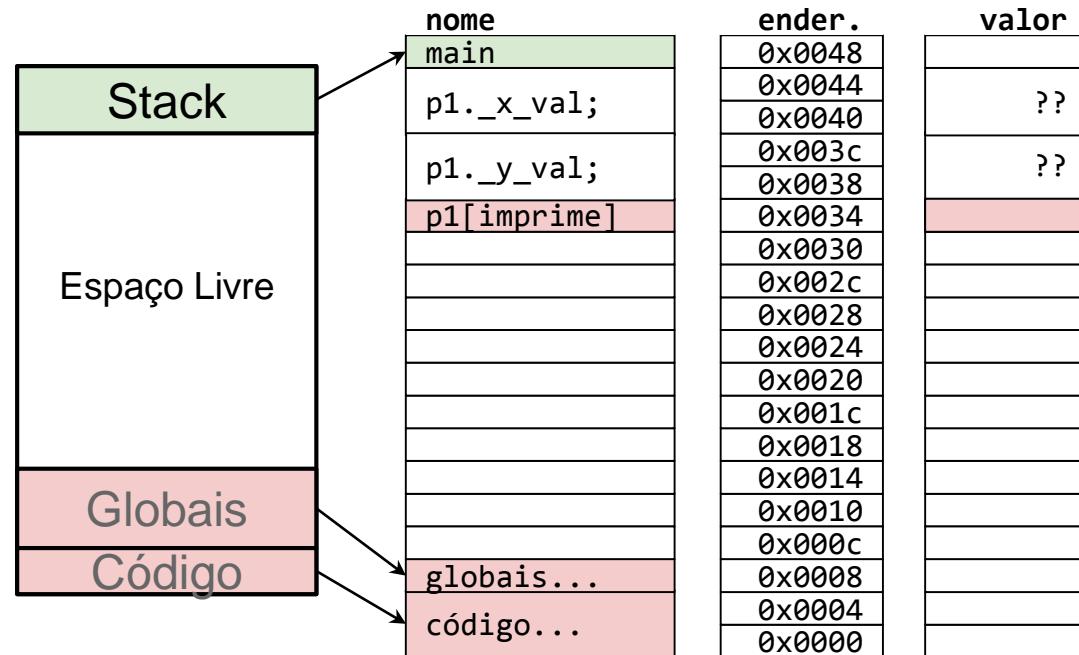
```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        // ...;  
    }  
};  
  
int main() {  
    Ponto p1;  
}
```



# Na memória (muito simplificada!!)

A máquina “guarda” que existe uma operação imprime no struct

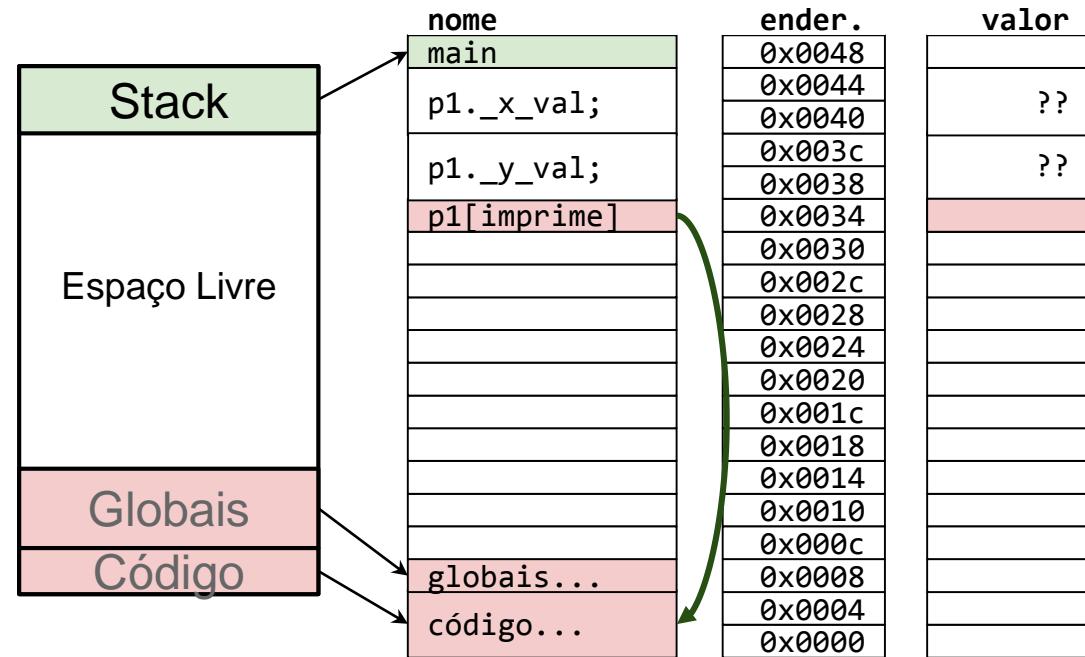
```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        // ...;  
    }  
};  
  
int main() {  
    Ponto p1;  
}
```



# Na memória (muito simplificada!!)

A mesma sabe exatamente qual código chamar

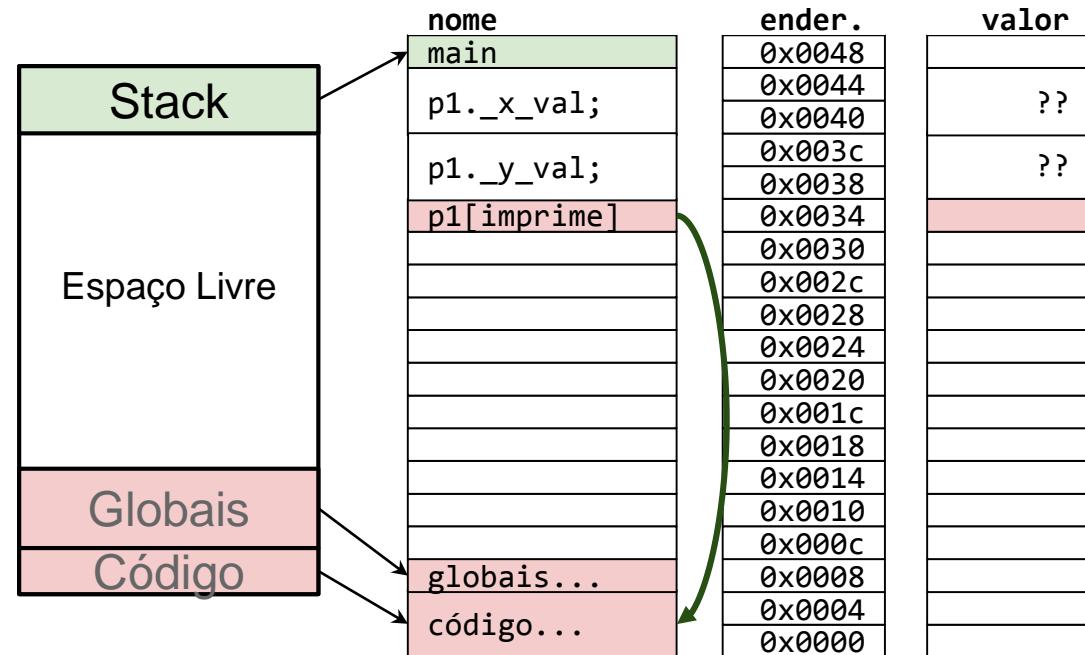
```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        // ...;  
    }  
};  
  
int main() {  
    Ponto p1;  
}
```



# Na memória (muito simplificada!!)

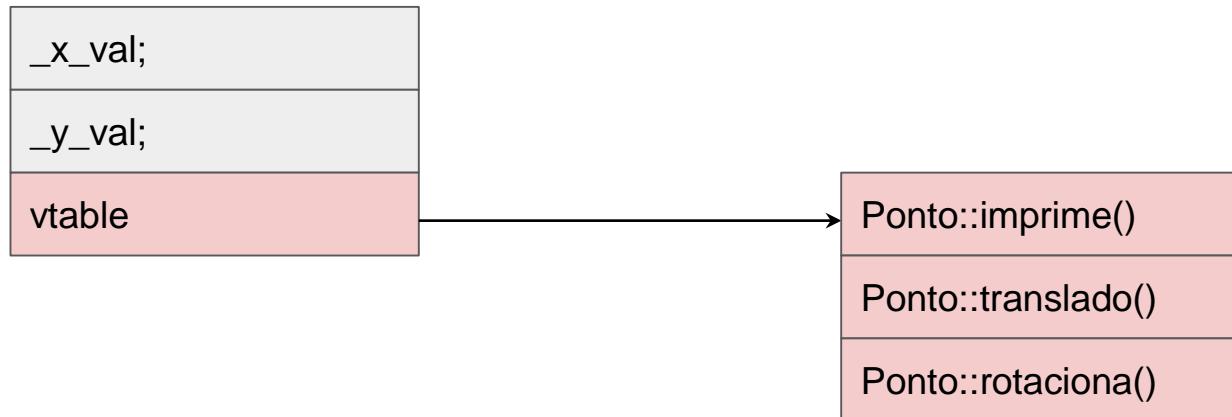
Se você conhece ponteiros para funções, mesma ideia.

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        // ...;  
    }  
};  
  
int main() {  
    Ponto p1;  
}
```



# A Tabela de Funções (vtable)

- Guarda funções que pertencem ao struct
- Como funciona uma chamada?



# Chamada de métodos

- Vamos usar o nome de método para uma função ou procedimento atrelado ao struct
- Facilita nossa transição para classes
- Vamos também simplificar o desenho da memória

# Chamada de métodos

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout <<  
        _x_val;  
    }  
};  
  
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
}
```

	_x_val	_y_val
main::p1 (stack)	??	??

# Chamada de métodos

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout <<  
        _x_val;  
    }  
};  
  
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
}
```

	_x_val	_y_val
main::p1 (stack)	??	??

# Chamada de métodos

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout <<  
        _x_val;  
    }  
};  
  
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
}
```



	_x_val	_y_val
main::p1 (stack)	2	??

# Chamada de métodos

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout <<  
        _x_val;  
    }  
};  
  
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
}  
D}
```



	_x_val	_y_val
main::p1 (stack)	2	3

# Qual o valor de \_x\_val?

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout <<  
        _x_val;  
    }  
};  
  
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
}
```



	<u>_x_val</u>	<u>_y_val</u>
main::p1 (stack)	2	3

# Qual o valor de \_x\_val? \_x\_val=2;

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operações  
    void imprime() {  
        std::cout <<  
        _x_val;  
    }  
};  
  
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
}
```



	<u>_x_val</u>	<u>_y_val</u>
main::p1 (stack)	2	3

# Chamada de métodos

- Ao chamar o método usamos a memória alocada naquele struct
- Eventualmente vamos chamar isto de objeto

# TADs e Estado

Em alto nível, é importante:

1. Lembrar que TADs são  
dados + operações
2. Dados são mutáveis, ou seja,  
guardam estado!
3. Aqui estamos fazendo isto com um struct.

# E agora?

```
int main() {  
    Ponto p1;  
    p1._x_val = 2;  
    p1._y_val = 3;  
    p1.imprime();  
    Ponto p2 = new Ponto;  
    p2->_x_val = 9;  
    p2->_y_val = 9;  
    p2->imprime();  
}
```

	_x_val	_y_val
<b>main::p1 (stack)</b>	2	3
<b>main::p2 (heap)</b>	9	9

# Chamada de métodos

- Cada chamada opera na memória que foi alocada
- Não importa se foi no stack ou no heap
- Em outras palavras, opera no estado

# Terminando o struct

Implementando as outras operações  
(ver código exemplo no github)

# Construtores

- Funções que iniciam o struct
- Chamadas de construtores

# Construtor

- Note que o mesmo não tem um valor de retorno? Qual o motivo?

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

Dados

# Construtor

- Note que o mesmo não tem um valor de retorno? Qual o motivo?

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	_x_val	_y_val
main::p1 (stack)	??	??

# Construtor

- Note que o mesmo não tem um valor de retorno? Qual o motivo?

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        →_x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	_x_val	_y_val
main::p1 (stack)	??	??

Dados

# Construtor

- Note que o mesmo não tem um valor de retorno? Qual o motivo?

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	_x_val	_y_val
main::p1 (stack)	2	??

Dados

# Construtor

- Note que o mesmo não tem um valor de retorno? Qual o motivo?

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```



	_x_val	_y_val
main::p1 (stack)	2	1

Dados

# Construtor

- Podemos chamar de duas formas possíveis

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	_x_val	_y_val
main::p1 (stack)	2	1

# Construtor

- Podemos chamar de duas formas possíveis

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        → _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	_x_val	_y_val
main::p1 (stack)	2	1
main::p2 (stack)	??	??

# Construtor

- Podemos chamar de duas formas possíveis

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	<u>_x_val</u>	<u>_y_val</u>
main::p1 (stack)	2	1
main::p2 (stack)	2	??

Dados

# Construtor

- Podemos chamar de duas formas possíveis

```
struct Ponto {  
    // Dados  
    double _x_val;  
    double _y_val;  
  
    // Operacoes  
    // 1. Construtor  
    Ponto(double x_val, double y_val) {  
        _x_val = x_val;  
        _y_val = y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
}
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1 (stack)</b>	2	1
<b>main::p2 (stack)</b>	2	3

Dados

# Pensando em um estado

- Temos que iniciar o estado
  - Construtor
- Depois
  - Podemos ler e escrever do mesmo

# Código Translação

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    void translado(Ponto &outro) {  
        _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1</b>	2	1
<b>main::p2</b>	2	3

# Note que o método opera em p1 e p2

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    →void translado(Ponto &outro) {  
        _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1
```

	_x_val	_y_val
main::p1	2	1
main::p2	2	3

# Note que o método opera em p1 e p2

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    void translado(Ponto &outro) {  
        → _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1</b>	2	1
<b>main::p2</b>	2	3

# Note que o método opera em p1 e p2

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    void translado(Ponto &outro) {  
        _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1</b>	4	1
<b>main::p2</b>	2	3

# Note que o método opera em p1 e p2

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    void translado(Ponto &outro) {  
        _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1</b>	4	4
<b>main::p2</b>	2	3

# Note que o método opera em p1 e p2

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    void translado(Ponto &outro) {  
        _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1  
x=4 y=4
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1</b>	4	4
<b>main::p2</b>	2	3

# Note que o método opera em p1 e p2

```
struct Ponto {  
    // .. ocultando o construtor e o imprime  
    // Operacoes  
    void translado(Ponto &outro) {  
        _x_val += outro._x_val;  
        _y_val += outro._y_val;  
    }  
};  
  
int main() {  
    Ponto p1 = Ponto(2, 1);  
    Ponto p2(2, 3);  
    p1.imprime();  
    p1.translado(p2);  
    p1.imprime();  
    p2.imprime();  
}
```

```
$ ./programa  
x=2 y=1  
x=4 y=4  
x=2 y=3
```

	<u>_x_val</u>	<u>_y_val</u>
<b>main::p1</b>	4	4
<b>main::p2</b>	2	3

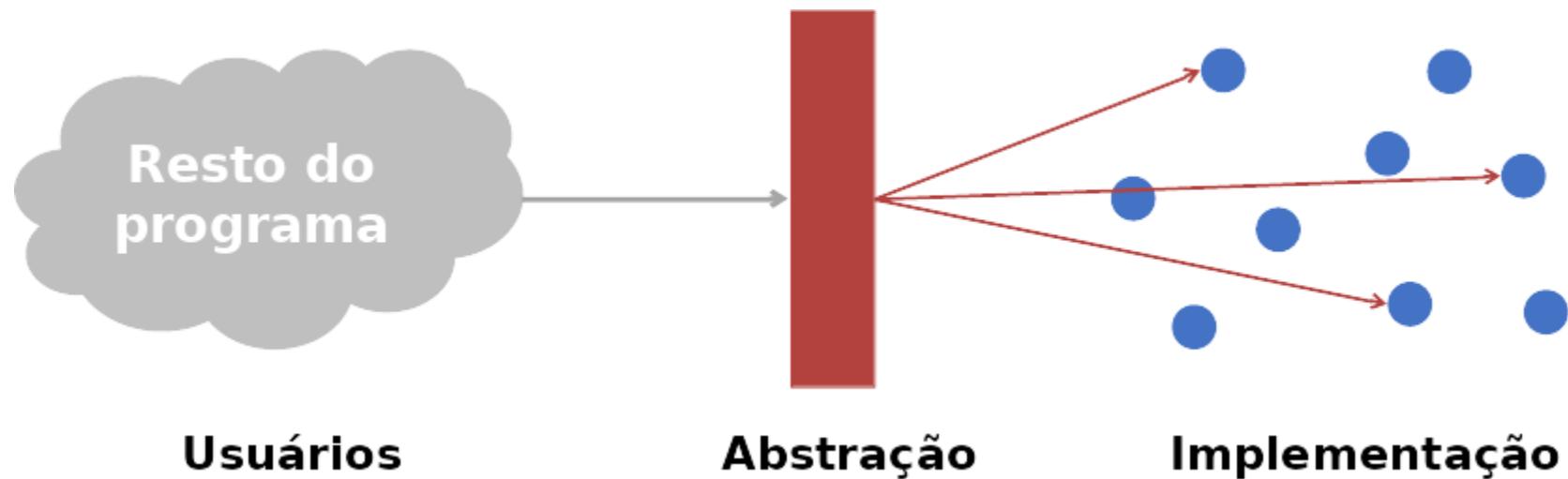
# Notas Finais

- Estamos iniciando o assunto da matéria
- Lembre-se de:
  - TADs → dados + operações
  - TADs guardam estado
  - Podemos usar um struct
    - Meio não definição

# Próxima Aula

- Criando TADs mais complexos
- Separando em módulos

# Aonde queremos chegar



Com TADs queremos que o resto do programa seja cliente. Apenas use as operações do mesmo.

# Exercícios

---

# **Como fazer um TAD aluno?**

---

# Primeiro problema

- Quais dados temos que representar?

# Primeiro problema

- Quais dados temos que representar?
  - Nome
  - Matrícula
  - Notas
- ...

# Segundo Problema

- Quais operações o aluno suporta?

# Segundo Problema

- Quais operações o aluno suporta?
  - Computar RSG

# **Como fazer um TAD matriz?**

---

# Primeiro problema

- Quais dados temos que representar?
  - células
  - n linhas
  - n colunas

# Segundo Problema

- Quais operações a matriz suporta?
  - Imprime
  - Soma
  - Multiplica
  - Liberar memória

# **Programação e Desenvolvimento de Software 2**

## **Boas práticas TAD (Modularização e Destrutores)**

---

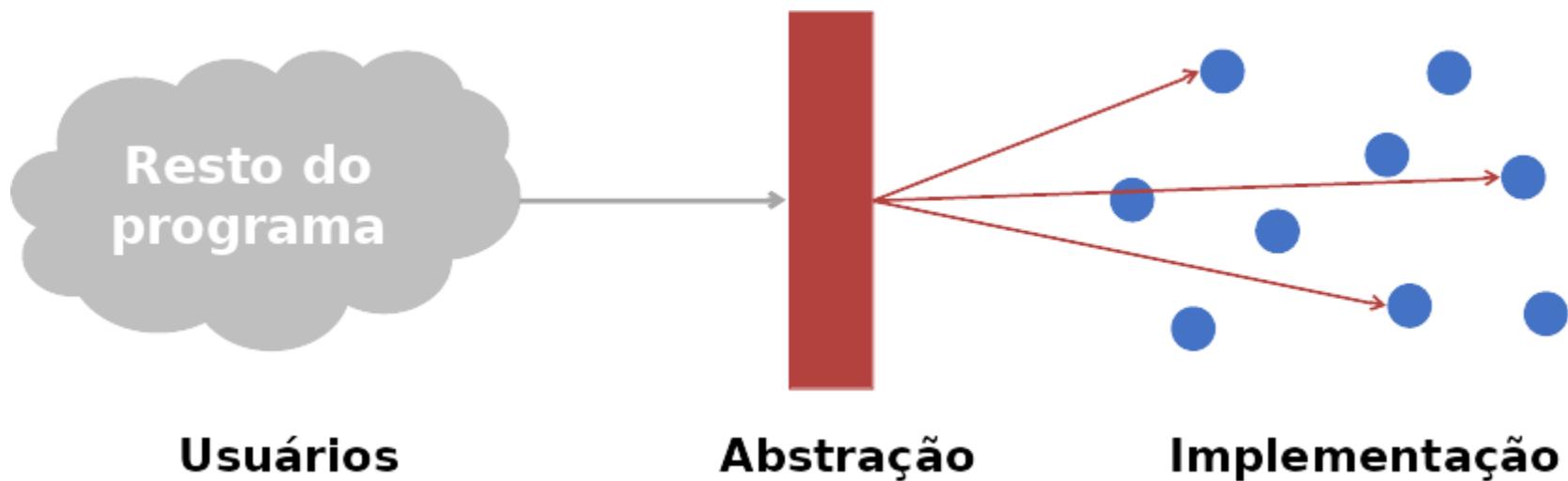
**Flavio Figueiredo**

<http://github.com/flaviovdf/programacao-2>

# **Projeto Modular**

---

# Lembrando do nosso objetivo



Com TADs queremos que o resto do programa seja cliente. Apenas use as operações do mesmo.

# Projeto Modular

## Propriedades

- Decomposição
- Composição
- Significado fechado
- Continuidade
- Proteção

# Projeto Modular

## Decomposição

### Nível de Projeto

Capaz de separar uma tarefa em subtarefas, que podem ser abordadas separadamente

### Nível de Software

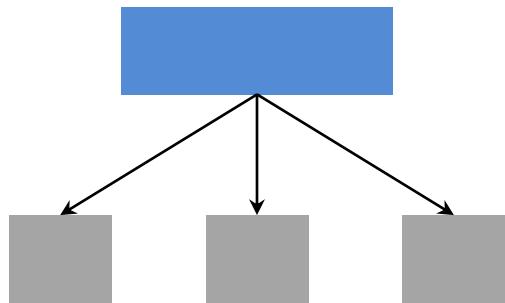
Capaz de trabalhar em cada um dos módulos do software independente do outros módulos

### O que pode prejudicar a decomposição?

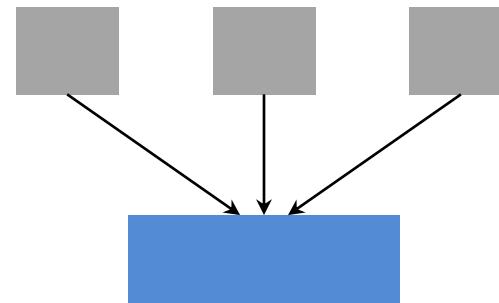
# Projeto Modular

## Composição

- Capacidade de conseguir combinar de forma livre diferentes elementos de software



Decomposição

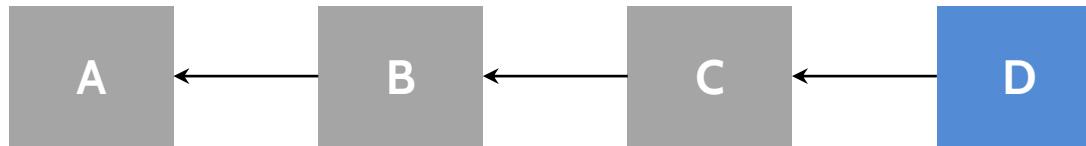


Composição

# Projeto Modular

## Significado fechado

- O programa deve ser compreendido por um leitor (usuário) que não possui acesso a outras (ou todas) partes do sistema



Problema: dependência sequencial.

# Projeto Modular

## Continuidade

- Alterações em parte da especificação demandam alterações em poucos módulos
- Bom exemplo
  - Utilização de constantes
- Mau exemplo
  - Dependência forte de um único módulo

# Projeto Modular

## Proteção

- Situações anormais em tempo de execução  
não são propagadas para outros módulos
  - Erros não detectados em outras partes
- Extensibilidade
- Validação dos dados nos módulos
  - Tipos, asserções, exceções

# **Modularizando um TAD simples**

---

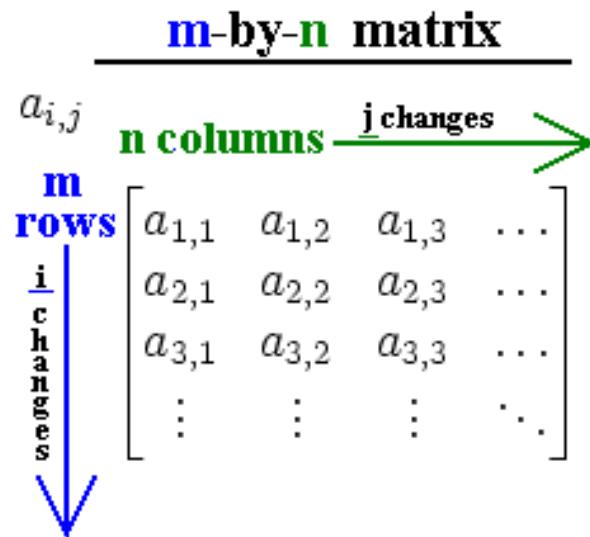
# Cabeçalhos

- Em C++, usamos arquivos de cabeçalhos
- Os mesmos descrevem os módulos

# Problema I

## Matriz

- Vamos criar um módulo matriz
- A mesma representa uma matriz que vai ser aloca dinamicamente



# Iniciando do .h

```
#ifndef PDS2_MATRIZ_H
#define PDS2_MATRIZ_H // Header guard, evitar erros

struct Matriz {
    // Dados
    int **_dados;
    int _n_linhas;
    int _n_colunas;

    // Construtor
    Matriz(int n_linhas, int n_colunas) // Note que não temos código
    // Destruitor
    ~Matriz(); // Já será explicado

    // Métodos
    void seta(int i, int j, int v); // M[i][j] = v
    int valor(int i, int j); // retorna valor i, j
    Matriz soma(Matriz &outra); // soma duas matrizes
};

#endif
```

# Código métodos seta e soma

```
#include <iostream>
#include "matriz.h"

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```

# Código métodos seta e soma

```
#include <iostream>
#include "matriz.h" ← Note o include do módulo matriz

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```

# Lendo 01: Retorne int

```
#include <iostream>
#include "matriz.h"

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```



A green arrow points from the word "Retorne" in the code to the "return" keyword in the "valor" method.

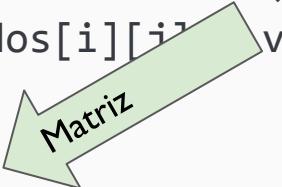
# Lendo 02: Na implementação do struct

```
#include <iostream>
#include "matriz.h"

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```



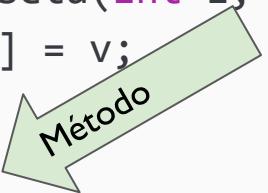
# Lendo 03: No método valor

```
#include <iostream>
#include "matriz.h"

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```



Método

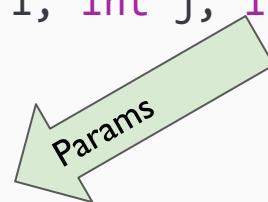
# Lendo 04: Que recebe i, j

```
#include <iostream>
#include "matriz.h"

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```



Params

# Lendo 04: Que recebe i, j

```
#include <iostream>
#include "matriz.h"

void Matriz::seta(int i, int j, int v) {
    _dados[i][j] = v;
}

int Matriz::valor(int i, int j) {
    return _dados[i][j];
}

Matriz Matriz::soma(Matriz &outra) {
    Matriz retorno = Matriz(_n_linhas, _n_colunas);
    for (int i = 0; i < _n_linhas; i++) {
        for (int j = 0; j < _n_colunas; j++) {
            retorno._dados[i][j] = valor(i, j) + outra.valor(i, j);
        }
    }
    return retorno;
}
```

# Arquivo main

```
#include <iostream>
#include "matriz.h"

int main(void) {
    Matriz m1(2, 2);
    Matriz m2(2, 2);

    std::cout << m1.valor(0, 0) << std::endl;
    std::cout << m2.valor(0, 0) << std::endl;

    m1.seta(0, 0, 1);
    std::cout << m1.valor(0, 0) << std::endl;

    m2.seta(0, 0, 2);
    std::cout << m2.valor(0, 0) << std::endl;

    Matriz m3 = m1.soma(m2);
    std::cout << m3.valor(0, 0) << std::endl;
}
```

# Arquivo main

- Faz uso dos módulos
- Não se preocupa como a matriz é implementada, cliente do módulo

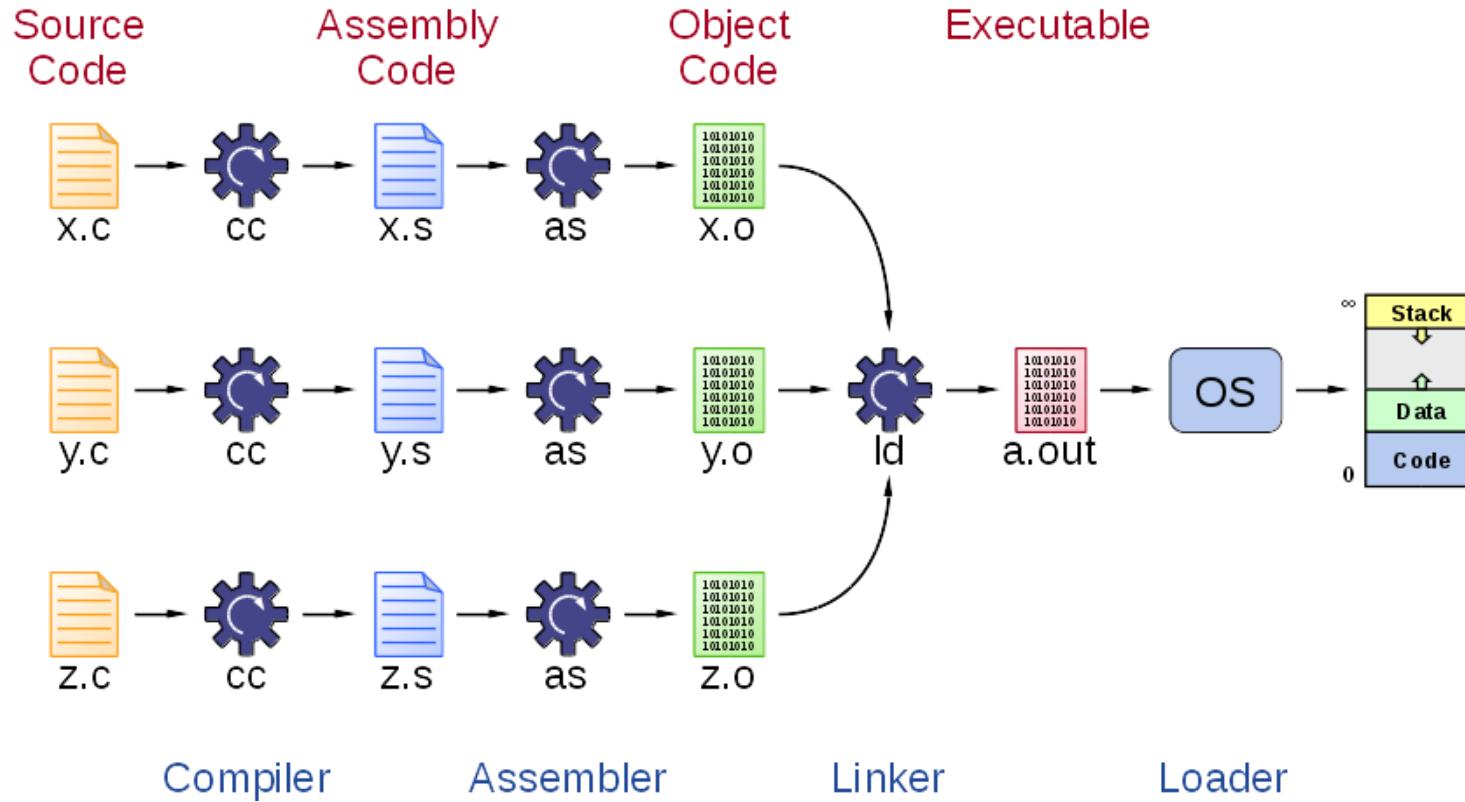
Note que não implementamos tudo  
aqui nos slides. Erro de compilação na prática. Estamos indo por partes!

# Compilando

```
$ g++ main.cpp matriz.cpp -o main
```

- Note que passamos dois arquivos
- O do main e o da matriz

# Compilação



# Construtores v. Destrutores

O nome diz tudo

- Procedimentos de inicialização
  - Usados apenas na criação de um novo objeto
- Procedimentos de destruição
  - Usados para liberar os recursos adquiridos na criação e utilizados por um certo objeto

# Construtores v. Destrutores

O nome diz tudo

- Destrutores tem um papel similar
- Liberar toda a memória que o objeto pode ter alocado
  - isto é, chamadas para **new**
- Também é útil para fechar recursos
  - Arquivos
  - Dentro outros

# Aquisição de Recurso é Inicialização

- Qual o motivo do destrutor?
- Uma boa prática é que todo objeto cuide da memória que o mesmo alocou
- Lembre-se do exemplo do banco
  - Em um sistema bem feito, apenas usamos os objetos

# Código Construtor e Destrutor

```
#include <iostream>
#include "matriz.h"

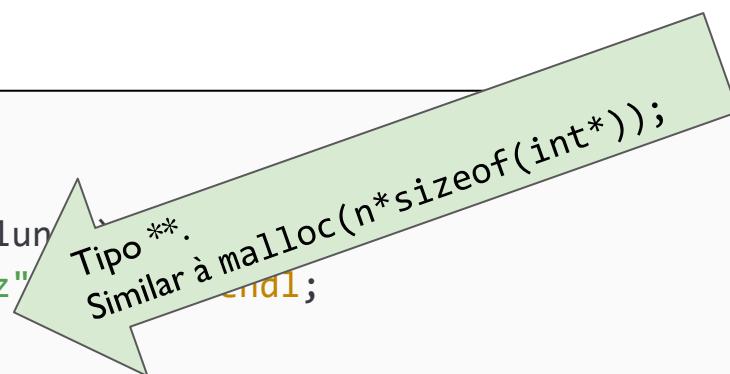
Matriz::Matriz(int n_linhas, int n_colunas) {
    std::cout << "Construindo uma matriz" << std::endl;
    _dados = new int*[n_linhas]();
    for (int i = 0; i < n_linhas; i++) {
        _dados[i] = new int[n_colunas];
    }
    _n_linhas = n_linhas;
    _n_colunas = n_colunas;
}

Matriz::~Matriz() {
    std::cout << "Destruindo uma matriz" << std::endl;
    for (int i = 0; i < _n_linhas; i++) {
        delete[] _dados[i];
    }
    delete[] _dados;
}
```

# Código

```
#include <iostream>
#include "matriz.h"
Matriz::Matriz(int n_linhas, int n_colunas) {
    std::cout << "Construindo uma matriz" << std::endl;
    _dados = new int*[n_linhas]();
    for (int i = 0; i < n_linhas; i++) {
        _dados[i] = new int[n_colunas];
    }
    _n_linhas = n_linhas;
    _n_colunas = n_colunas;
}

Matriz::~Matriz() {
    std::cout << "Destruindo uma matriz" << std::endl;
    for (int i = 0; i < _n_linhas; i++) {
        delete[] _dados[i];
    }
    delete[] _dados;
}
```



Tipo \*\*.  
Similar à malloc(n\*sizeof(int\*));

# Exemplificando Destruidores

```
#include "matriz.h"

int main(void) {
    Matriz *matriz = new Matriz(100, 100);
    delete matriz;

    Matriz matriz2(100, 100);
    return 0;
}
```



```
$ ./main
```

# Exemplificando Destruidores

```
#include "matriz.h"

int main(void) {
    Matriz *matriz = new Matriz(100, 100);
    delete matriz;

    Matriz matriz2(100, 100);
    return 0;
}
```



```
$ ./main
Alocando matriz
```

# Exemplificando Destruidores

```
#include "matriz.h"

int main(void) {
    Matriz *matriz = new Matriz(100, 100);
    delete matriz;


    Matriz matriz2(100, 100);
    return 0;
}
```

```
$ ./main
Construindo matriz
Destruindo matrix
```

# Exemplificando Destruidores

```
#include "matriz.h"

int main(void) {
    Matriz *matriz = new Matriz(100, 100);
    delete matriz;

    Matriz matriz2(100, 100);
    return 0;
}
```



```
$ ./main
Construindo matriz
Destruindo matriz
Construindo matriz
```

# Exemplificando Destruidores

```
#include "matriz.h"

int main(void) {
    Matriz *matriz = new Matriz(100, 100);
    delete matriz;

    Matriz matriz2(100, 100);
    return 0;
}
```



```
$ ./main
Construindo matriz
Destruindo matriz
Construindo matriz
Destruindo matriz
```

# Destruidores

São chamados sempre que o objeto é desalocado

- Destruidores são chamados tanto para:
  - Objetos no heap
    - Depois de um delete
  - Objetos no stack
    - Depois que a função termina

# Destruidores

São chamados sempre que o objeto é desalocado

Lembrando que

- O computador cuida do stack
- Você cuida do heap
- Por isso fazemos o destrutor, a matriz é  
alocada dinamicamente no heap!

# Listas

---

# Listas Lineares

- Sequência de zero ou mais itens
  - $x_1, x_2, \dots, x_n$ , na qual  $x_i$
- Posições relativas
  - Assumindo  $n \geq 1$ ,  $x_1$  é o primeiro item da lista e  $x_n$  é o último item da lista.
  - $x_i$  precede  $x_{i+1}$  para  $i = 1, 2, \dots, n - 1$
  - $x_i$  sucede  $x_{i-1}$  para  $i = 2, 3, \dots, n$
  - $x_i$  é dito estar na  $i$ -ésima posição da lista

# Tipos Abstratos de Dados (TADs)

## Lista de números inteiros

- Considere uma lista de inteiros. Poderíamos definir TAD Lista, com as seguintes operações:
  - faça a lista vazia;
  - obtenha o primeiro elemento da lista; se a lista estiver vazia, então retorna nulo;
  - insira um elemento na lista.

# Tipos Abstratos de Dados (TADs)

## Lista de números inteiros

- Quais outras operações podem ser definidas?
  - Retirar o i-ésimo item.
  - Localizar o i-ésimo item
  - Fazer uma cópia da lista linear.
  - Pesquisar a ocorrência de um item com um valor particular em algum componente.

# Solução Zero

```
#ifndef PDS2_LISTA_VETOR_H
#define PDS2_LISTA_VETOR_H
#define TAMANHO 100 // Constante no .h. Em C++ também existe o const

struct ListaVetorInteiros {
    // Dados
    int *_elementos; // Vetor de elementos que será alocado dinâmicamente (heap)
    int _num_elementos_inseridos;

    // Construtor
    ListaVetorInteiros();
    // Destruitor
    ~ListaVetorInteiros();
    // Insere um inteiro na lista
    void inserir_elemento(int elemento);
    // Imprime a lista
    void imprimir();
};

#endif
```

# **Como Implementar?**

---

# Construtor (e início do .h)

```
#include <iostream>           Em algum momento vamos imprimir a lista
#include "listavetor.h"

ListaVetorInteiros::ListaVetorInteiros() {
    _elementos = new int[TAMANHO]();
    _num_elementos_inseridos = 0;
}
```

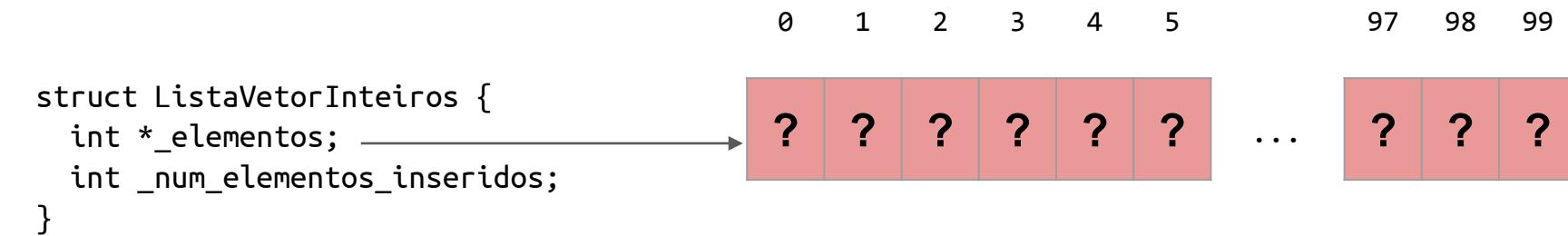
# Construtor (e início do .h)

```
ListaVetorInteiros::ListaVetorInteiros() {  
    _elementos = new int[TAMANHO]();  
    _num_elementos_inseridos = 0;  
}
```

```
struct ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos;  
}
```

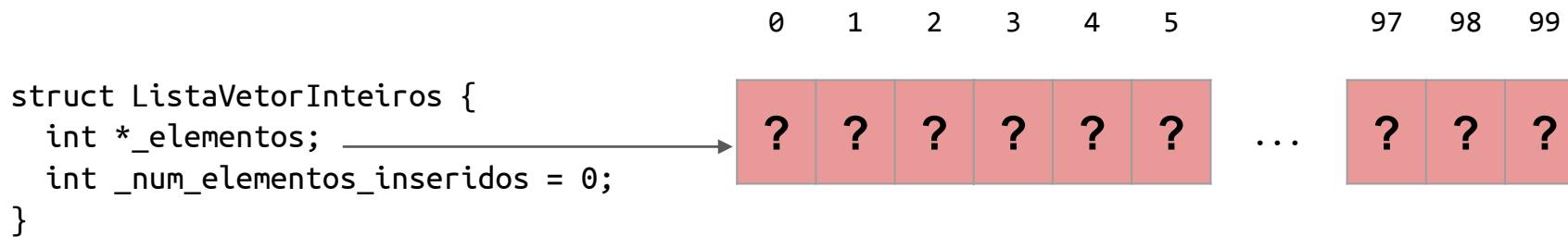
# Construtor (e início do .h)

```
ListaVetorInteiros::ListaVetorInteiros() {  
    _elementos = new int[TAMANHO]();  
    _num_elementos_inseridos = 0;  
}
```



# Construtor (e início do .h)

```
ListaVetorInteiros::ListaVetorInteiros() {  
    _elementos = new int[TAMANHO]();  
    _num_elementos_inseridos = 0;  
}
```

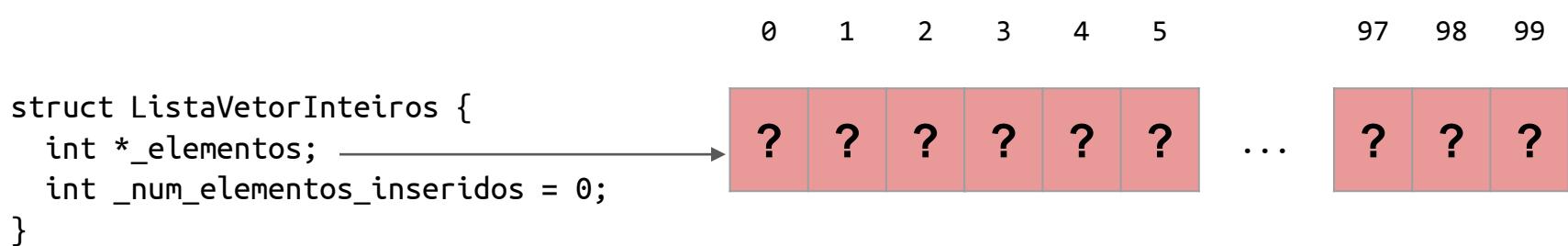


# **Como adicionar elementos?**

---

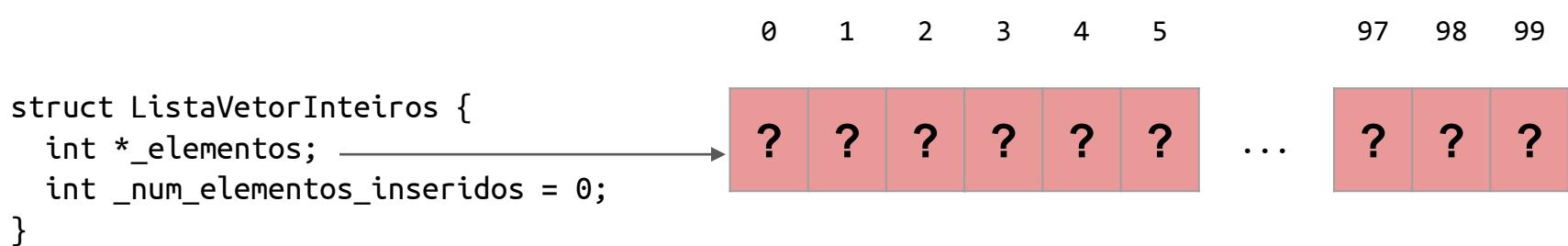
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



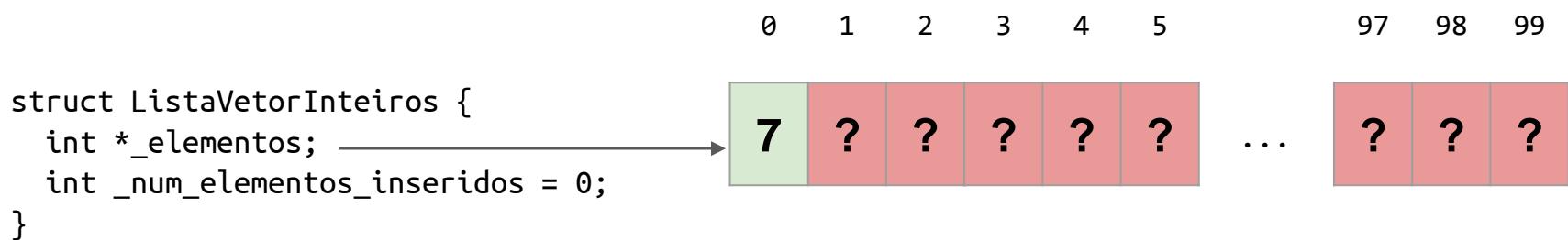
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



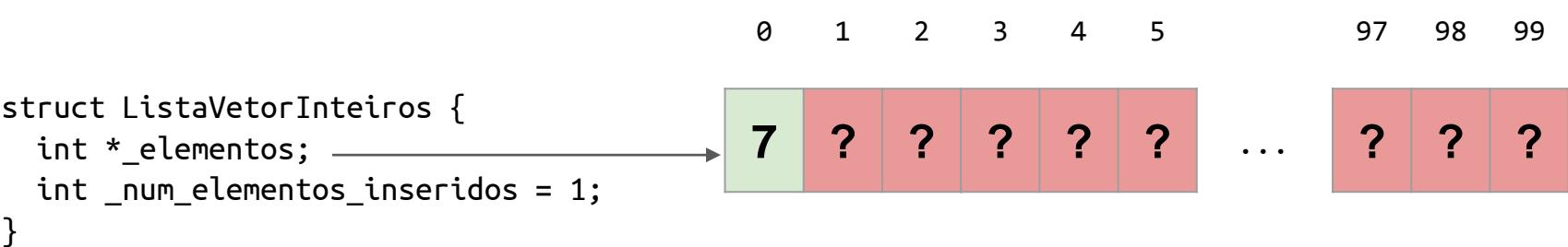
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



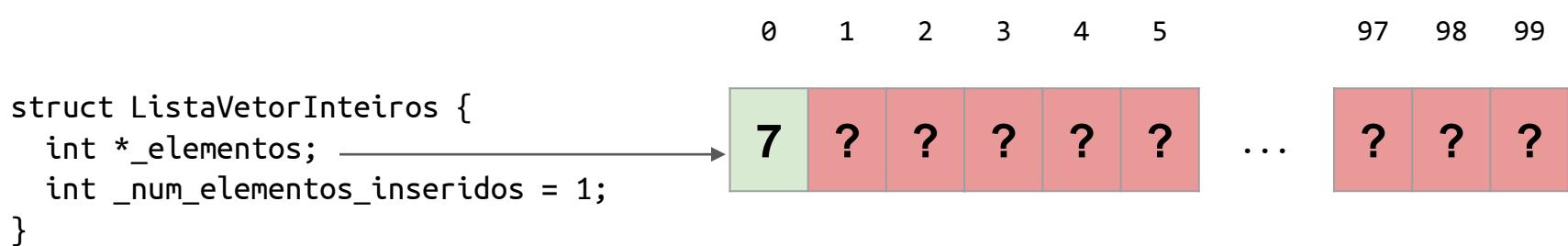
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



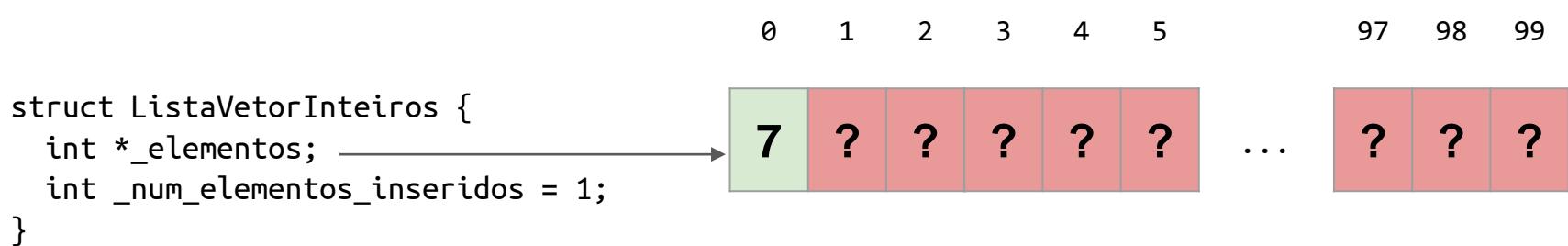
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



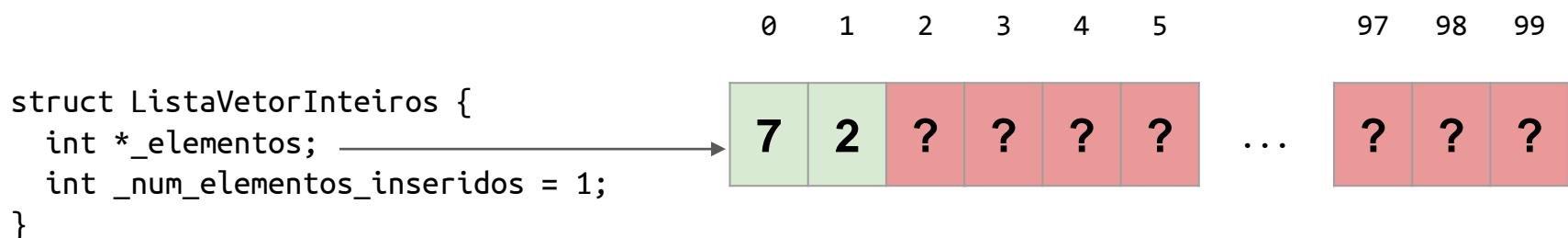
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



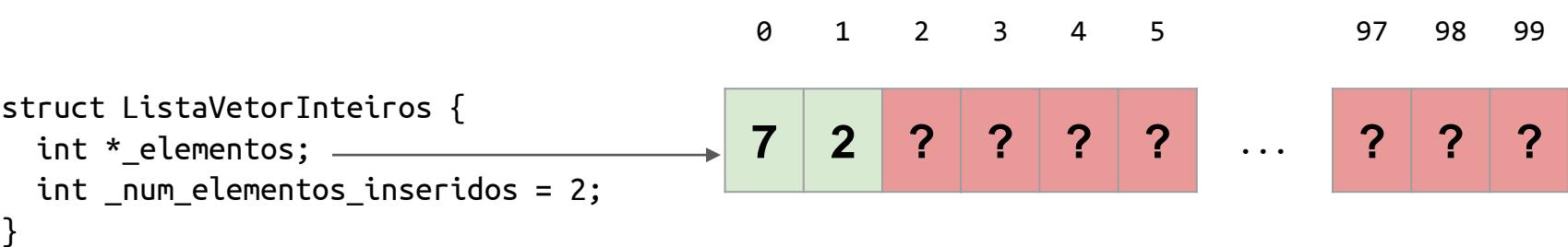
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



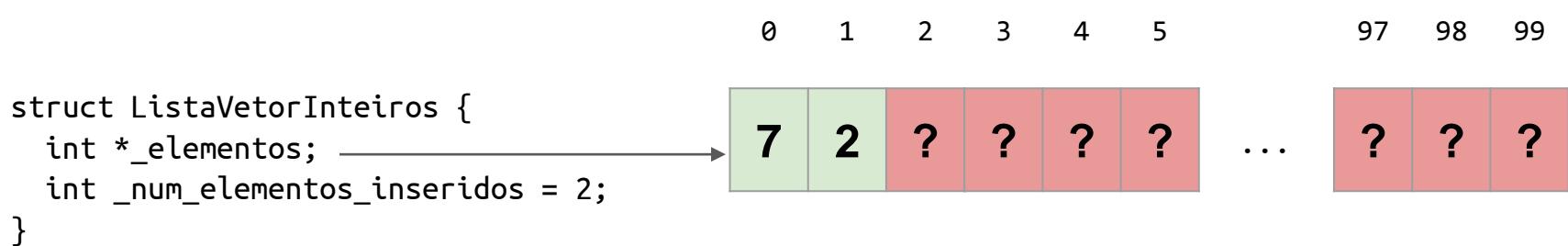
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



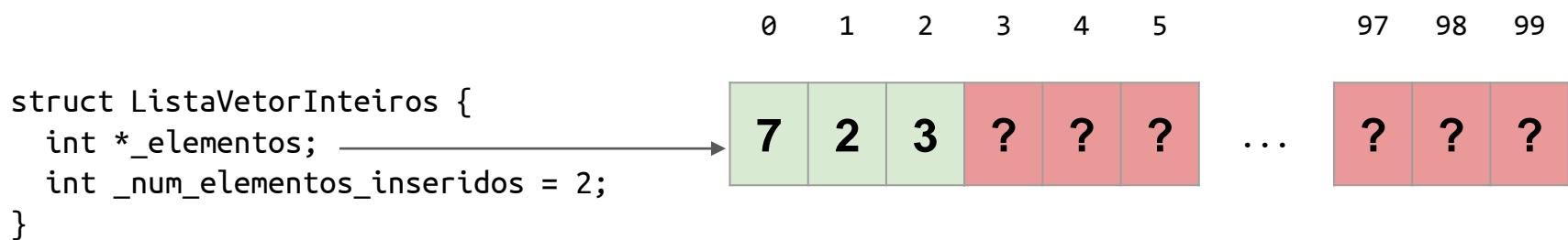
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



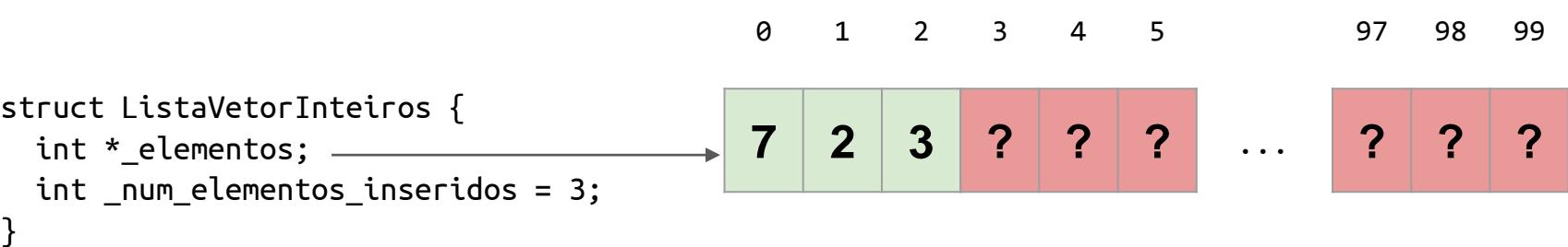
# Adicionando elementos?

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



# E por aí vai...

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == TAMANHO) {  
        std::cerr << "Erro, lista cheia" << std::endl;  
        exit(1);  
    }  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```



## □ Trivial

```
void ListaVetorInteiros::imprimir() {
    for (int i = 0; i < _num_elementos_inseridos; i++)
        std::cout << _elementos[i] << " ";
    std::cout << std::endl;
}
```

# Destruitor

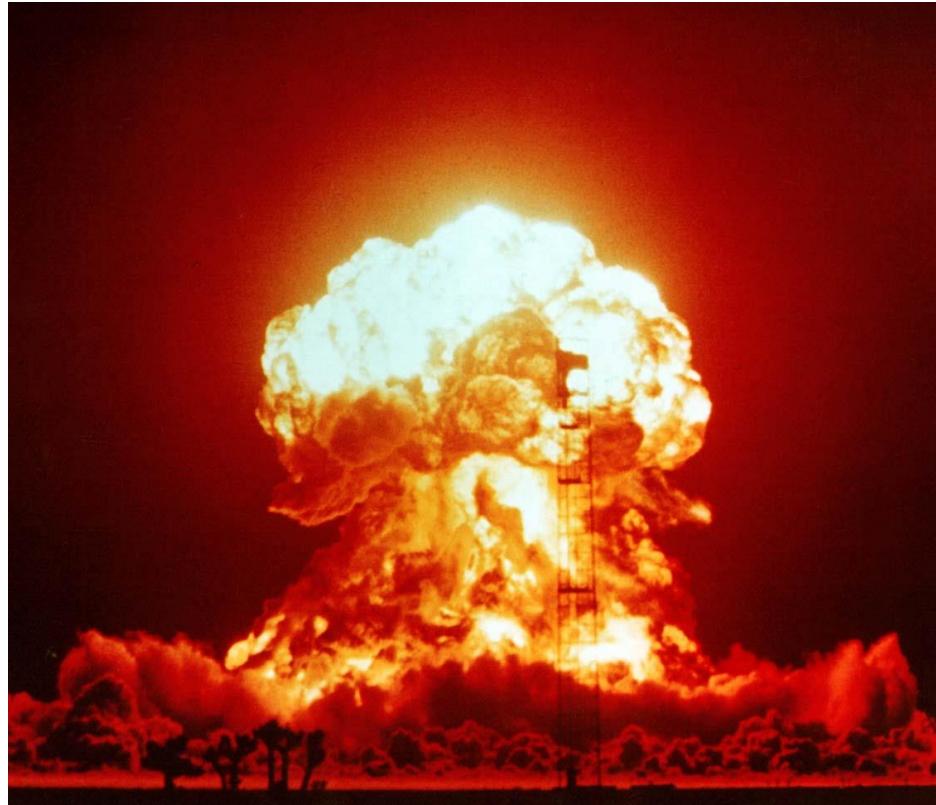
- Alocamos um vetor
  - I **new**
- Precisamos de I **delete**
- Lembrando, cada **new** → I **delete**

```
ListaVetorInteiros::~ListaVetorInteiros() {  
    delete[] _elementos;  
}
```

# **Mais de 100 elementos?**

---

# Mais de 100 elementos?



```
#ifndef PDS2_LISTA_VETOR_H
#define PDS2_LISTA_VETOR_H

#define TAMANHO_INICIAL 100 // Tamanho inicial que será aumentado

struct ListaVetorInteiros {
    // Dados
    int *_elementos;
    int _num_elementos_inseridos;
    int _capacidade; // Tamanho atual que também será aumentado

    // Construtor
    ListaVetorInteiros();
    // Destrutor
    ~ListaVetorInteiros();
    // Insere um inteiro na lista
    void inserir_elemento(int elemento);
    // Imprime a lista
    void imprimir();
};

#endif
```

# Construtor



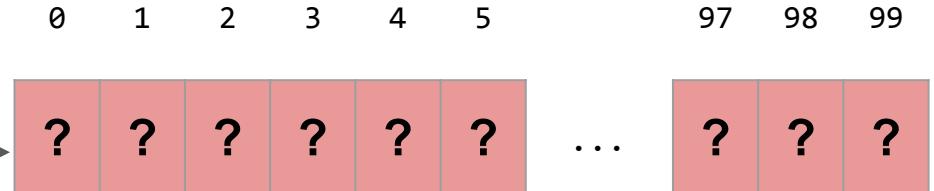
```
ListaVetorInteiros::ListaVetorInteiros() {  
    _elementos = new int[TAMANHO_INICIAL]();  
    _num_elementos_inseridos = 0;  
    _capacidade = TAMANHO_INICIAL;  
}
```

# Constructor

```
ListaVetorInteiros::ListaVetorInteiros() {  
    _elementos = new int[TAMANHO_INICIAL]();  
    _num_elementos_inseridos = 0;  
    _capacidade = TAMANHO_INICIAL;  
}
```



```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 0;  
    int _capacidade = 100;  
}
```



# Métodos que não mudam

- Imprime
- Destrutor

# Complicação

## Inserir elemento

- Inserir elemento
- Caso o vetor fique cheio
  - Duplicar o mesmo
  - Copiar tudo para o novo
  - Aumentar a capacidade
- Estamos implementando o **vector**
  - Nome do container na **STL**

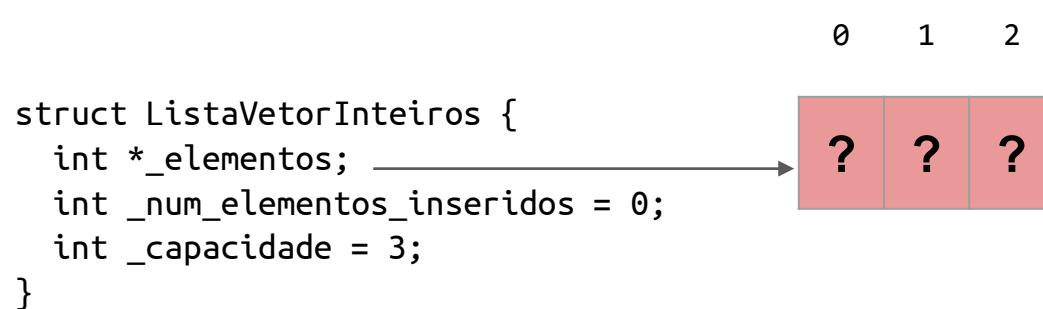
# Inserir elemento

```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    // . . .  
    _elementos[_num_elementos_inseridos] = elemento;  
    _num_elementos_inseridos++;  
}
```

# Passo a Passo

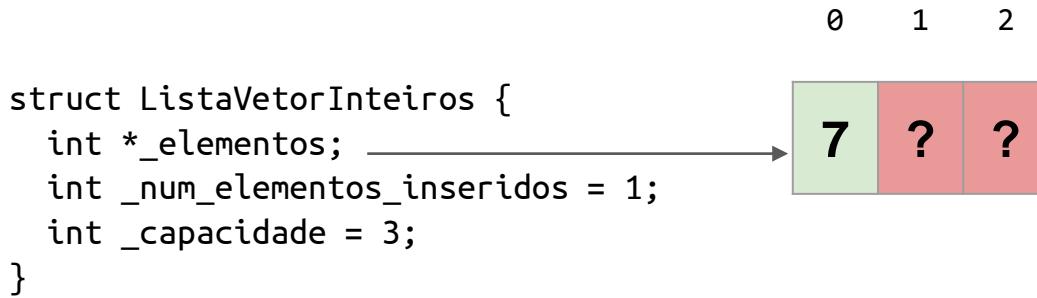
Alocamos tamanho inicial.

Vamos supor que seja igual a 3



# Passo a Passo

## Inserindo um elemento



# Passo a Passo

## Outro

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 2;  
    int _capacidade = 3;  
}
```



# Passo a Passo

+ |

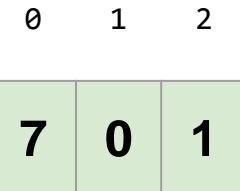
```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



# Passo a Passo

+ outro?!

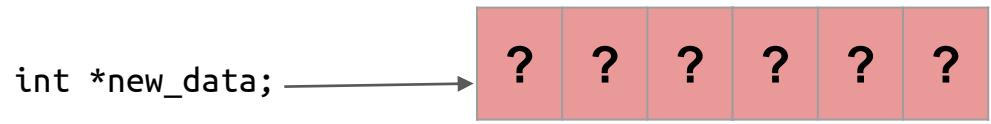
```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



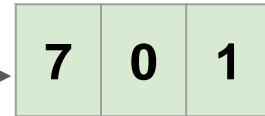
# Passo a Passo

+ alocamos  
espaço

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 0;  
    int _capacidade = 3;  
}
```



0    1    2

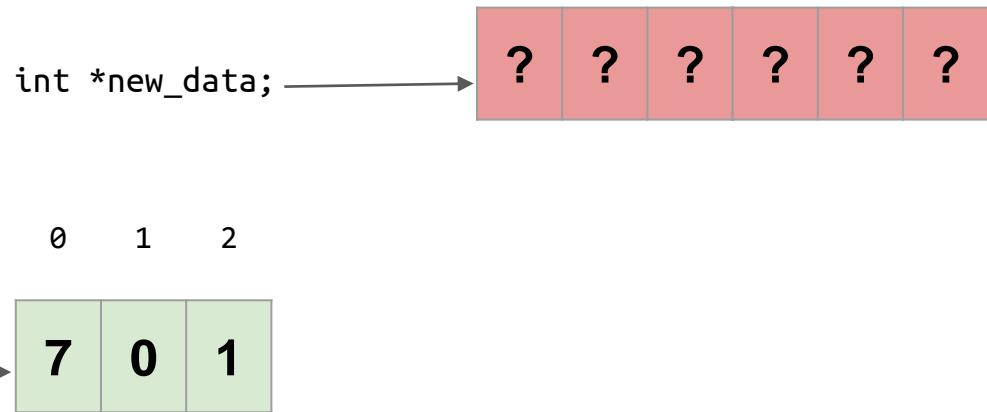


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == _capacidade) {  
        int *new_data = new int[_capacidade * 2];  
  
        for (int i = 0; i < _num_elementos_inseridos; i++)  
            new_data[i] = _elementos[i];  
  
        delete[] _elementos;  
        _elementos = new_data;  
        _capacidade = _capacidade * 2;  
    }  
    // . . .  
}
```

# Passo a Passo

+ copiamos os dados

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```

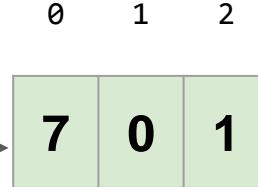
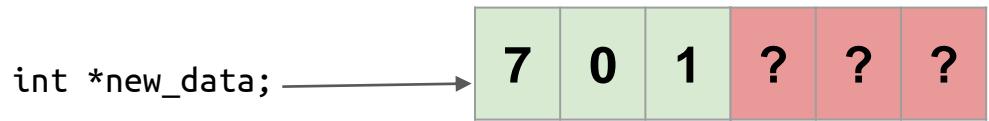


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == _capacidade) {  
        int *new_data = new int[_capacidade * 2];  
  
        for (int i = 0; i < _num_elementos_inseridos; i++)  
            new_data[i] = _elementos[i];  
  
        delete[] _elementos;  
        _elementos = new_data;  
        _capacidade = _capacidade * 2;  
    }  
    // . . .  
}
```

# Passo a Passo

+ copiamos os dados

```
struct ListaVetorInteiros {  
    int *_elementos; ——————→  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == _capacidade) {  
        int *new_data = new int[_capacidade * 2];  
  
        for (int i = 0; i < _num_elementos_inseridos; i++)  
            new_data[i] = _elementos[i];  
  
        delete[] _elementos;  
        _elementos = new_data;  
        _capacidade = _capacidade * 2;  
    }  
    // . . .  
}
```

# Passo a Passo

+ apagamos os dados antigos

```
struct ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == _capacidade) {  
        int *new_data = new int[_capacidade * 2];  
  
        for (int i = 0; i < _num_elementos_inseridos; i++)  
            new_data[i] = _elementos[i];  
  
        delete[] _elementos;  
        _elementos = new_data;  
        _capacidade = _capacidade * 2;  
    }  
    // . . .  
}
```

# Passo a Passo

+ colocamos os novos no local

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```

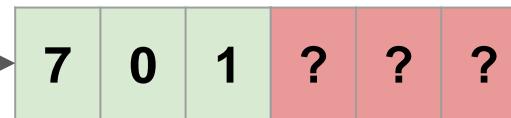


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == _capacidade) {  
        int *new_data = new int[_capacidade * 2];  
  
        for (int i = 0; i < _num_elementos_inseridos; i++)  
            new_data[i] = _elementos[i];  
  
        delete[] _elementos;  
        _elementos = new_data;  
        _capacidade = _capacidade * 2;  
    }  
    // . . .  
}
```

# Passo a Passo

+ aumentamos a capacidade

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 6;  
}
```

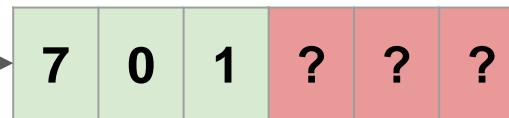


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    if (_num_elementos_inseridos == _capacidade) {  
        int *new_data = new int[_capacidade * 2];  
  
        for (int i = 0; i < _num_elementos_inseridos; i++)  
            new_data[i] = _elementos[i];  
  
        delete[] _elementos;  
        _elementos = new_data;  
        _capacidade = _capacidade * 2;  
    }  
    // . . .  
}
```

# Passo a Passo

## Agora estamos igual a antes

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 6;  
}
```

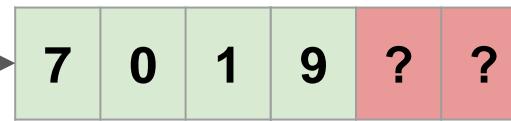


```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    // . . .  
    _elementos[_num_elementos_inserido] = elemento;  
    _num_elementos_inseridos++;  
}
```

# Passo a Passo

## Agora estamos igual a antes

```
struct ListaVetorInteiros {  
    int *_elementos; _____  
    int _num_elementos_inseridos = 4;  
    int _capacidade = 6;  
}
```



```
void ListaVetorInteiros::inserir_elemento(int elemento) {  
    // . . .  
    _elementos[_num_elementos_inserido] = elemento;  
    _num_elementos_inseridos++;  
}
```

# E para remover o último elemento?

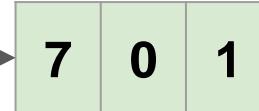
- Quero uma nova operação no meu TAD
  - Atualizar .h
  - Implementar no .cpp
- Remover o último elemento

# E para remover o último elemento?

- Quero uma nova operação no meu TAD
- Remover o último elemento

```
void ListaVetorInteiros::remover_ultimo() {  
    _num_elementos_inseridos--;  
}
```

```
struct ListaVetorInteiros {  
    int *_elementos; ——————>  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



# E para remover o último elemento?

- Quero uma nova operação no meu TAD
- Remover o último elemento

```
void ListaVetorInteiros::remover_ultimo() {  
    _num_elementos_inseridos--;  
}
```

```
struct ListaVetorInteiros {  
    int *_elementos; ——————> 7 | 0 | 1  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```

# E para remover o último elemento?

- Quero uma nova operação no meu TAD
- Remover o último elemento

```
void ListaVetorInteiros::remover_ultimo() {  
    _num_elementos_inseridos--;  
}
```



```
struct ListaVetorInteiros {  
    int *_elementos; ——————>  
    int _num_elementos_inseridos = 3;  
    int _capacidade = 3;  
}
```



# Removendo o Primeiro Elemento

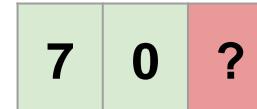
- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Funciona bem?

# Removendo o Primeiro Elemento

- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Chato não desperdiçar memória

```
void ListaVetorInteiros::remover_primeiro() {  
    _inicio++;  
    _num_elementos_inseridos--;  
}
```

```
struct ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 2;  
    int _capacidade = 3;  
    int _inicio = 0;  
}
```



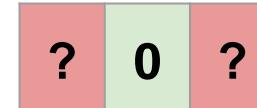
# Removendo o Primeiro Elemento

- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Chato não desperdiçar memória

```
void ListaVetorInteiros::remover_primeiro() {  
    _inicio++;  
    _num_elementos_inseridos--;  
}
```



```
struct ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 2;  
    int _capacidade = 3;  
    int _inicio = 1;  
}
```

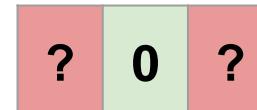


# Removendo o Primeiro Elemento

- Podemos copiar a ideia anterior
- Guardar um índice para o início
- Chato não desperdiçar memória

```
void ListaVetorInteiros::remover_primeiro() {  
    _inicio++;  
    _num_elementos_inseridos--;  
}
```

```
struct ListaVetorInteiros {  
    int *_elementos;  
    int _num_elementos_inseridos = 1;  
    int _capacidade = 3;  
    int _inicio = 1;  
}
```



# Programação e Desenvolvimento de Software 2

## Listas encadeadas

---

Flavio Figueiredo

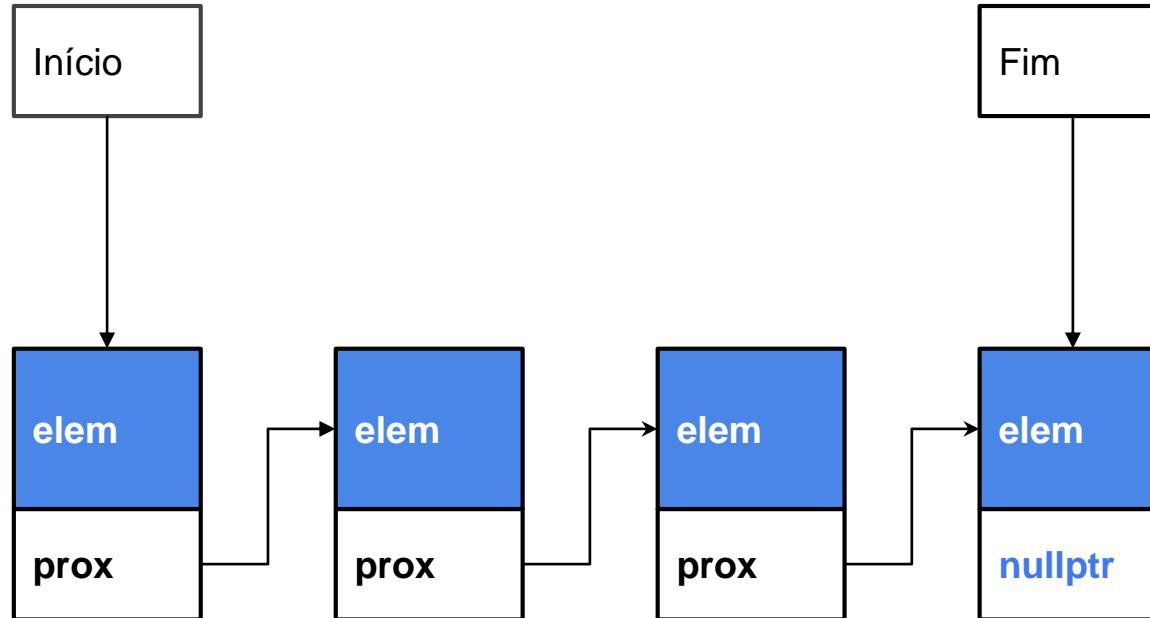
<http://github.com/flaviovdf/programacao-2>

# **Listas com Ponteiros**

---

# Lista com Ponteiros

- Para casos complicados com arrays
- Podemos explorar ponteiros



# Cabeçalho

```
#ifndef PDS2_NODE_H
#define PDS2_NODE_H
struct node_t {
    int elemento;
    node_t *proximo;
};

struct ListaSimplesmenteEncadeada {
    node_t *_inicio;           ← Início e fim da lista
    node_t *_fim;
    int _num_elementos_inseridos;

    ListaSimplesmenteEncadeada();
    ~ListaSimplesmenteEncadeada();
    void inserir_elemento(int elemento);
    void imprimir();
};
#endif
```

Struct sem métodos. Representa um elemento

Início e fim da lista

# Iniciamos Assim

```
struct node_t {  
    int elemento;  
    node_t *next;  
};
```



?

nullptr!

# Ficamos Assim

```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```



```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```

23

```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```

# Ficamos Assim

```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```



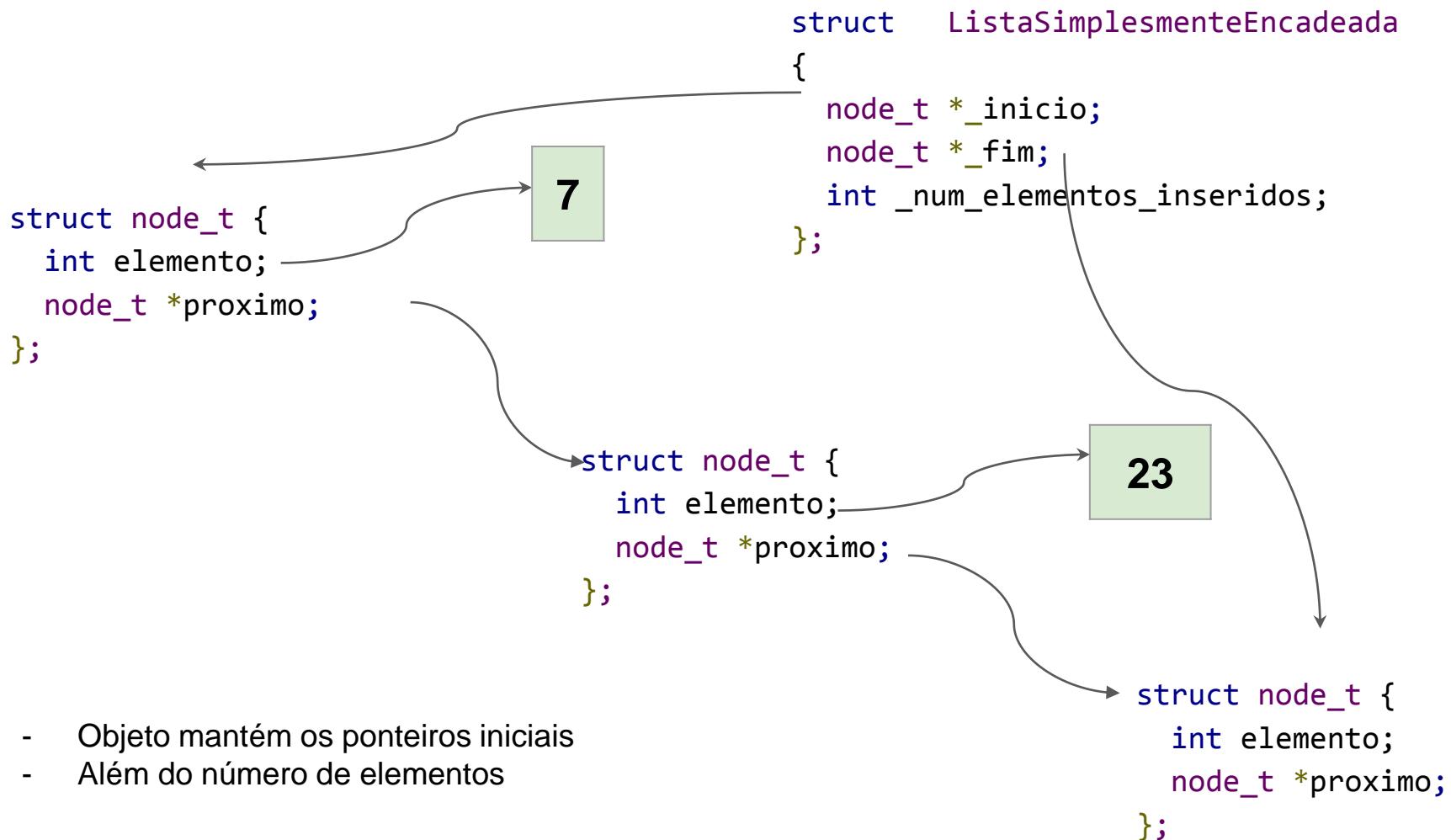
```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```

```
23
```

```
struct node_t {  
    int elemento;  
    node_t *proximo;  
};
```

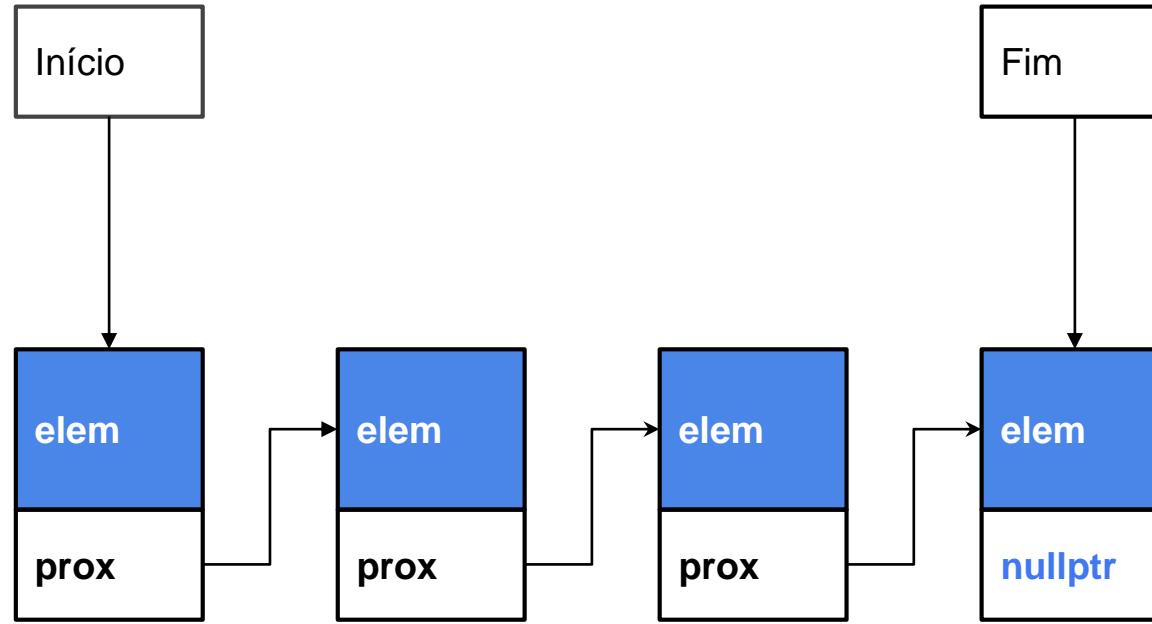
- Um struct aponta para outro
- Cada Struct mantém um valor

# Ficamos Assim



- Objeto mantém os ponteiros iniciais
- Além do número de elementos

# Mais Abstrato



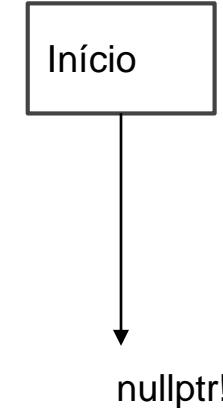
# Vamos Implementar?

# Construtor

```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    _inicio = nullptr;  
    _fim = nullptr;  
    _num_elementos_inseridos = 0;  
}
```

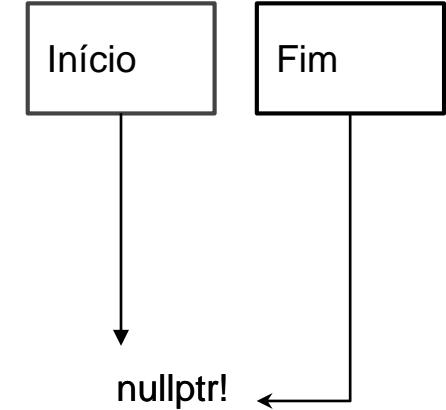
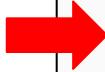
# Construtor

```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    _inicio = nullptr;  
    _fim = nullptr;  
    _num_elementos_inseridos = 0;  
}
```



# Construtor

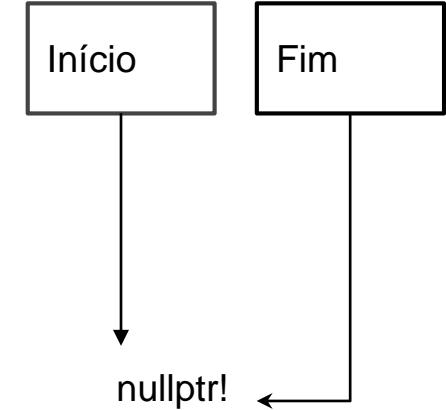
```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    _inicio = nullptr;  
    _fim = nullptr;  
    _num_elementos_inseridos = 0;  
}
```



# Construtor

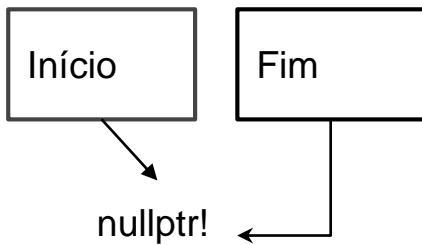
```
ListaSimplesmenteEncadeada::ListaSimplesmenteEncadeada() {  
    _inicio = nullptr;  
    _fim = nullptr;  
    _num_elementos_inseridos = 0;  
}
```

```
Objeto ListaVetorInteiros {  
    node_t *inicio = nullptr;  
    node_t *fim = nullptr;  
    int _num_elementos_inseridos = 0;  
}
```



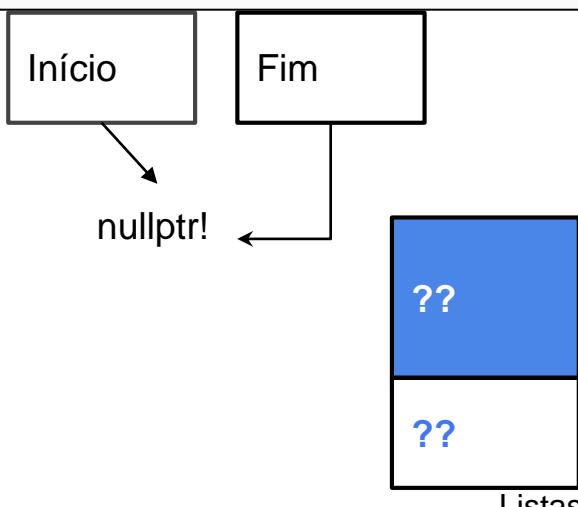
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



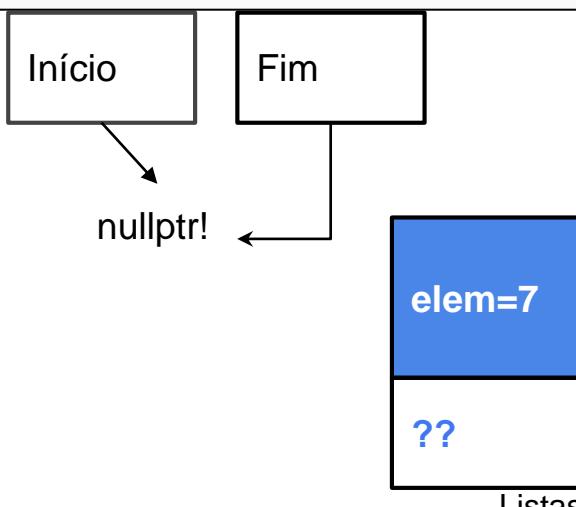
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



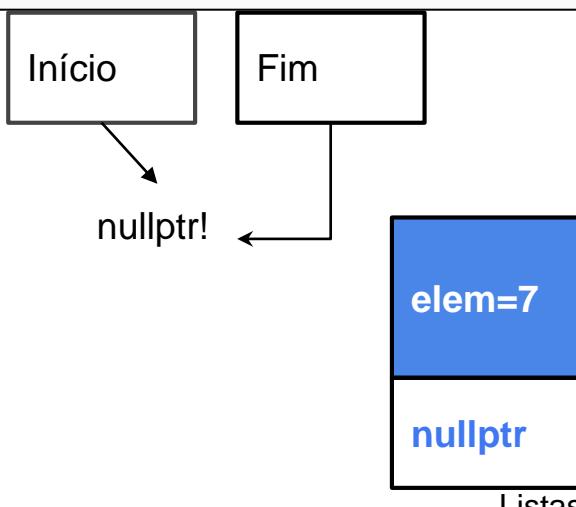
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



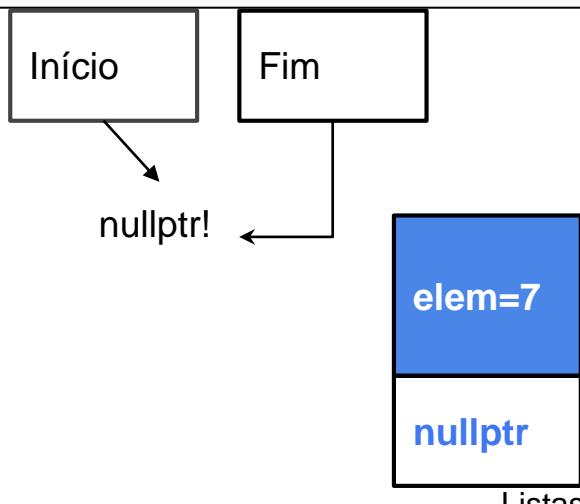
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



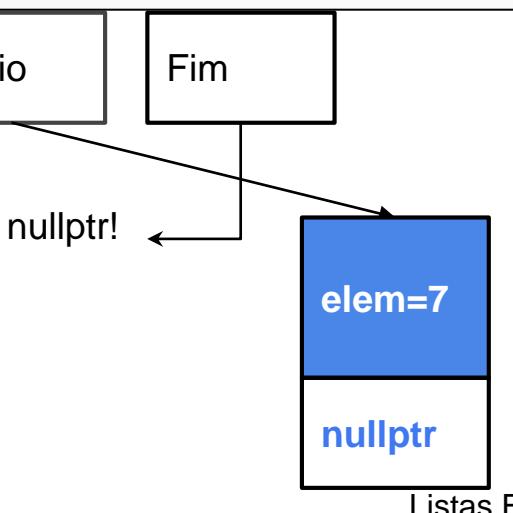
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



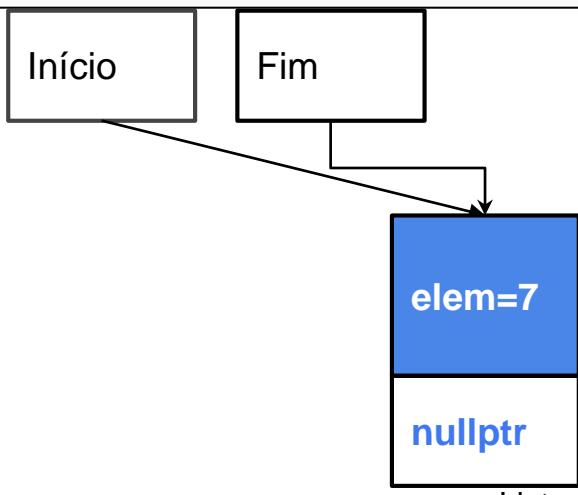
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



# Adicionando

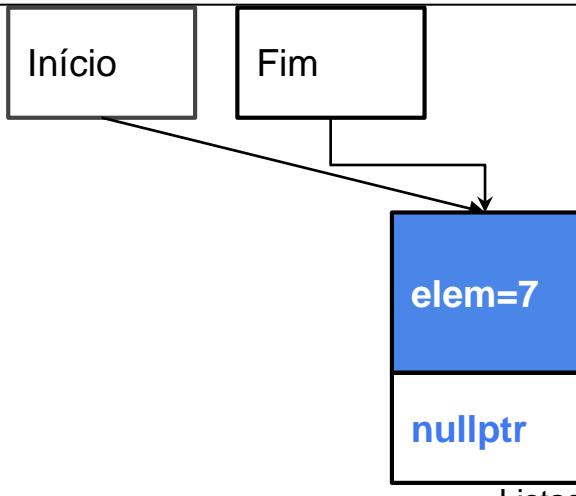
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



Listas Encadeadas

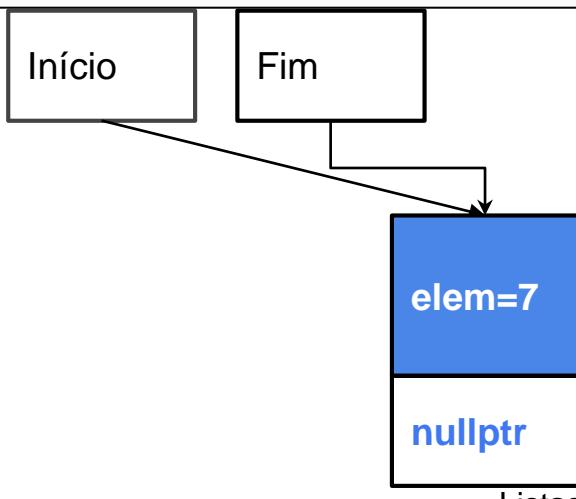
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



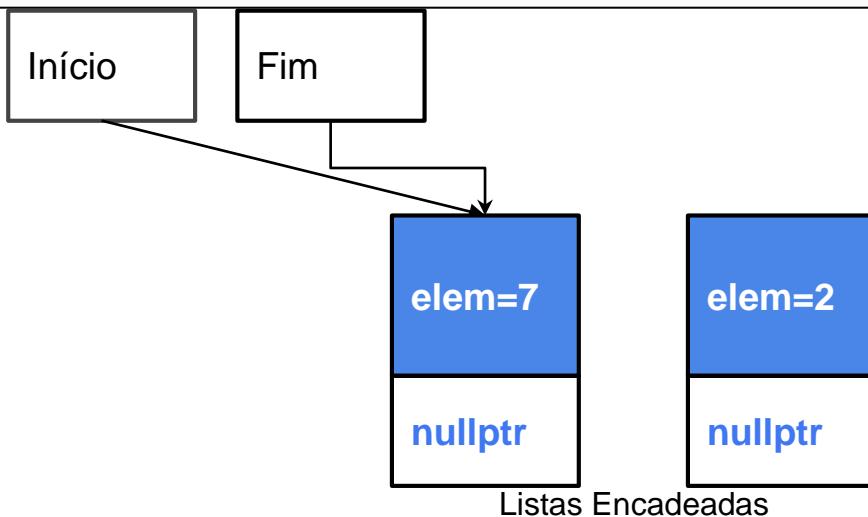
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



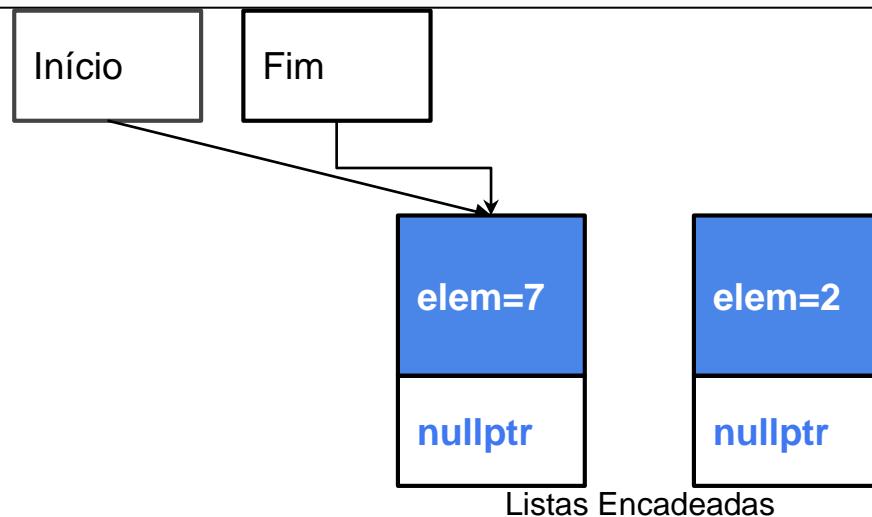
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



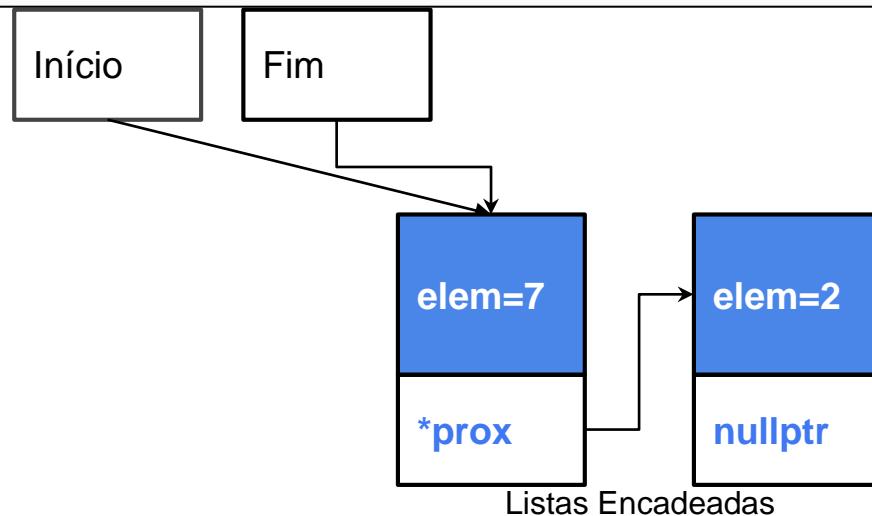
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->proxima = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->proxima = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



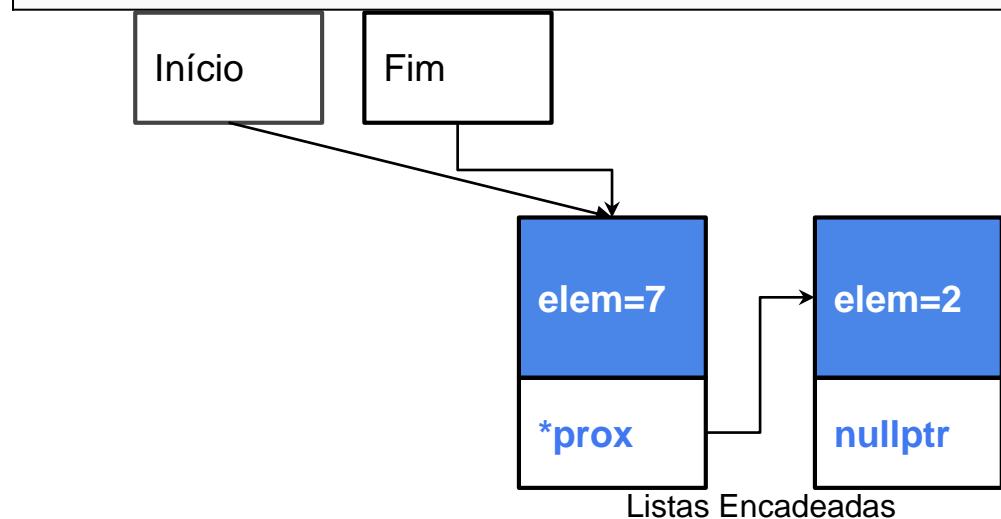
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->prox = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->prox = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



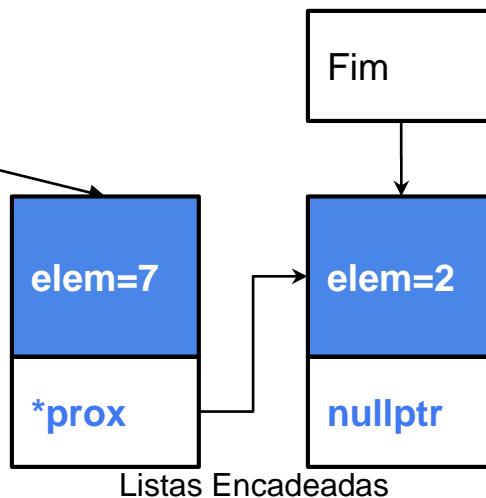
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->prox = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->prox = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



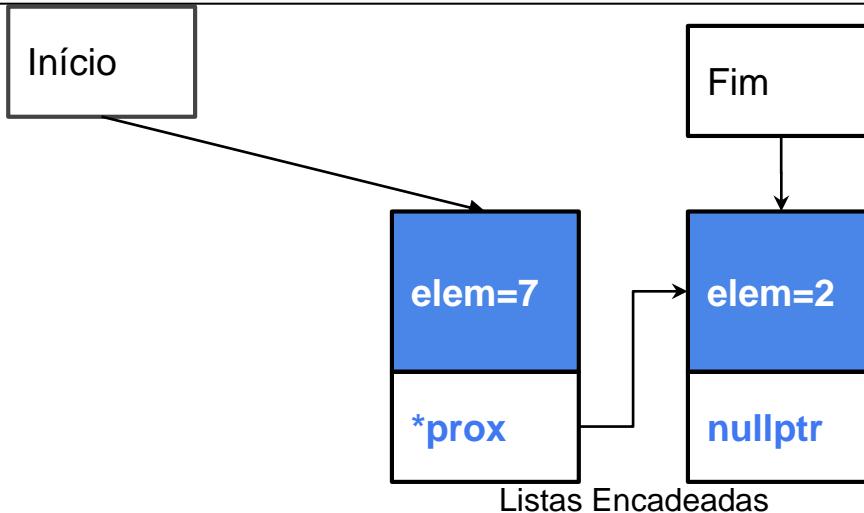
# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->prox = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->prox = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```

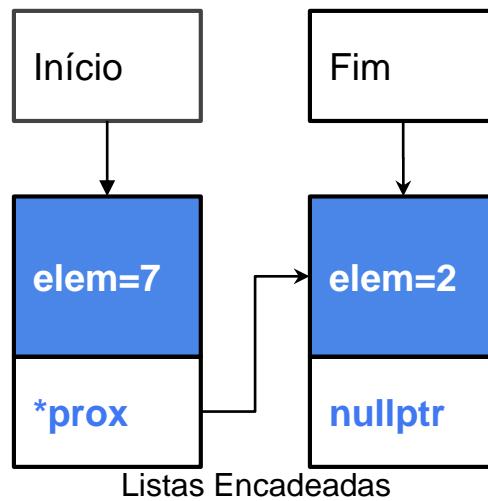


# Adicionando

```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->prox = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->prox = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



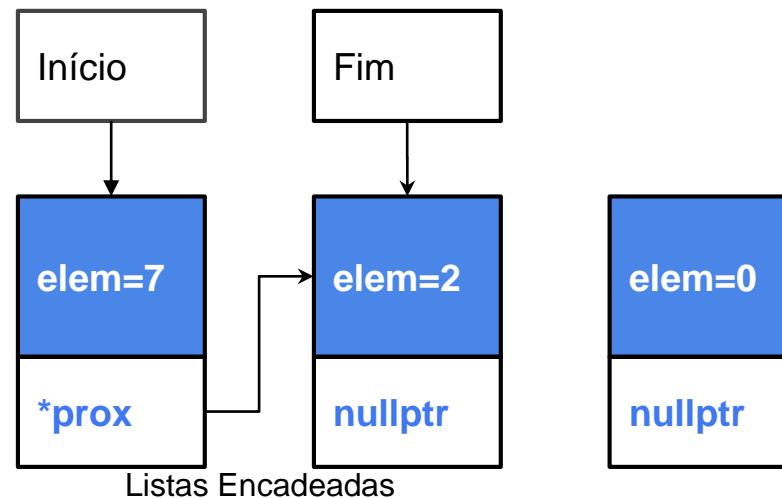
```
void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {  
    node_t *novo = new node_t();  
    novo->elemento = elemento;  
    novo->prox = nullptr;  
    if (_inicio == nullptr) {  
        _inicio = novo;  
        _fim = novo;  
    } else {  
        _fim->prox = novo;  
        _fim = novo;  
    }  
    _num_elementos_inseridos++;  
}
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->prox = nullptr;
    if (_inicio == nullptr) {
        _inicio = novo;
        _fim = novo;
    } else {
        _fim->prox = novo;
        _fim = novo;
    }
    _num_elementos_inseridos++;
}

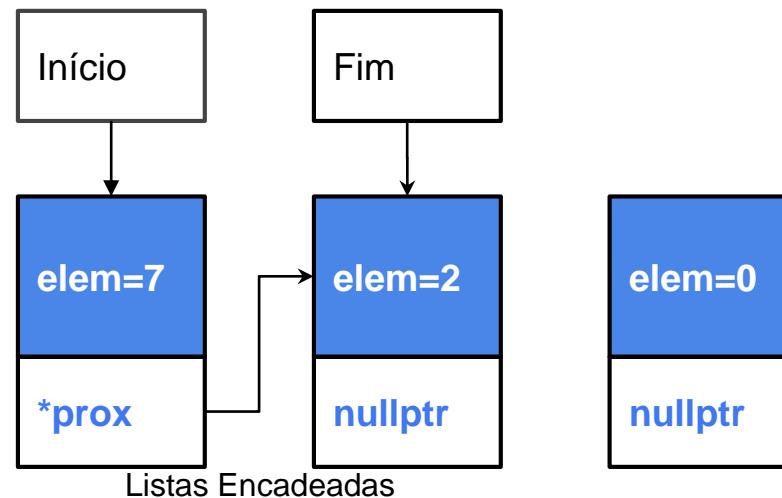
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->prox = nullptr;
    if (_inicio == nullptr) {
        _inicio = novo;
        _fim = novo;
    } else {
        → _fim->prox = novo;
        _fim = novo;
    }
    _num_elementos_inseridos++;
}

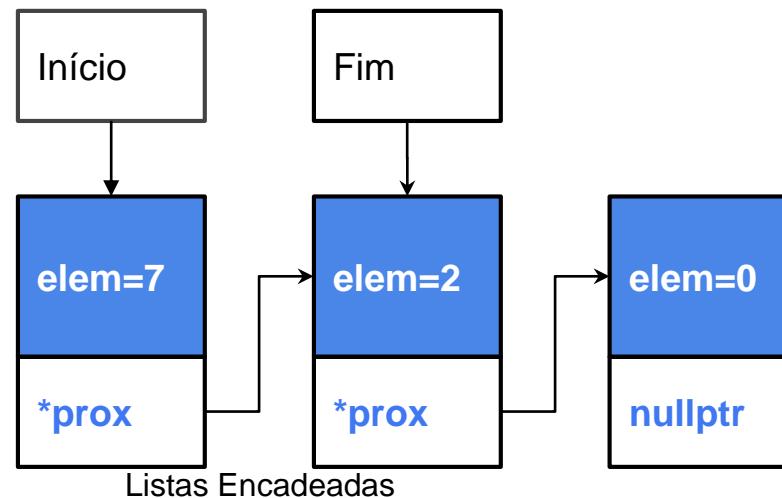
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->prox = nullptr;
    if (_inicio == nullptr) {
        _inicio = novo;
        _fim = novo;
    } else {
        _fim->prox = novo;
        _fim = novo;
    }
    _num_elementos_inseridos++;
}

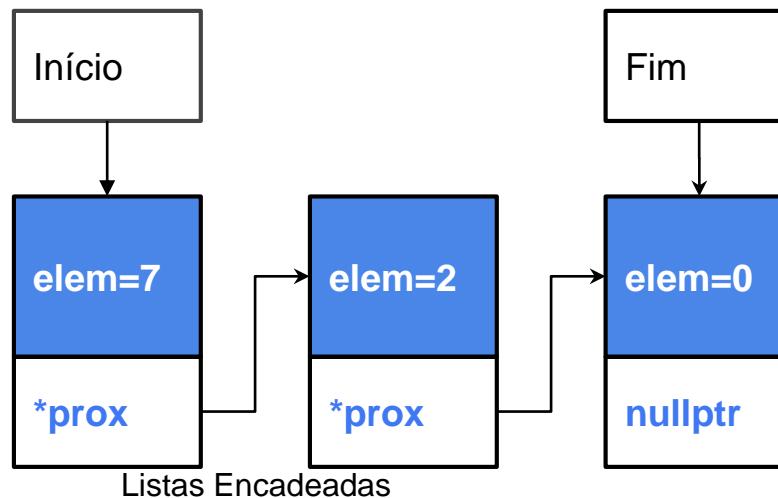
```



```

void ListaSimplesmenteEncadeada::inserir_elemento(int elemento) {
    node_t *novo = new node_t();
    novo->elemento = elemento;
    novo->prox = nullptr;
    if (_inicio == nullptr) {
        _inicio = novo;
        _fim = novo;
    } else {
        _fim->prox = novo;
        _fim = novo;
    }
    _num_elementos_inseridos++;
}

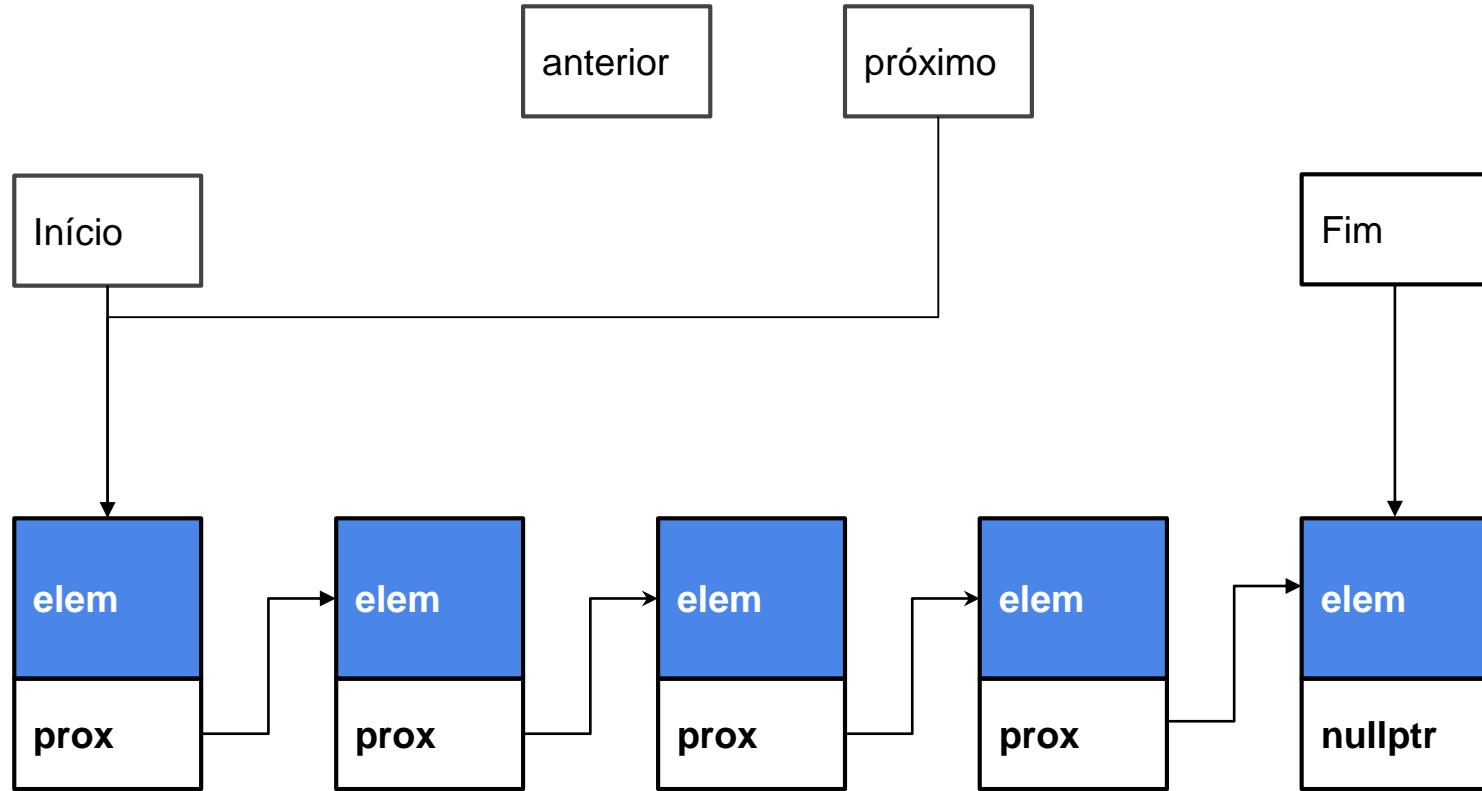
```



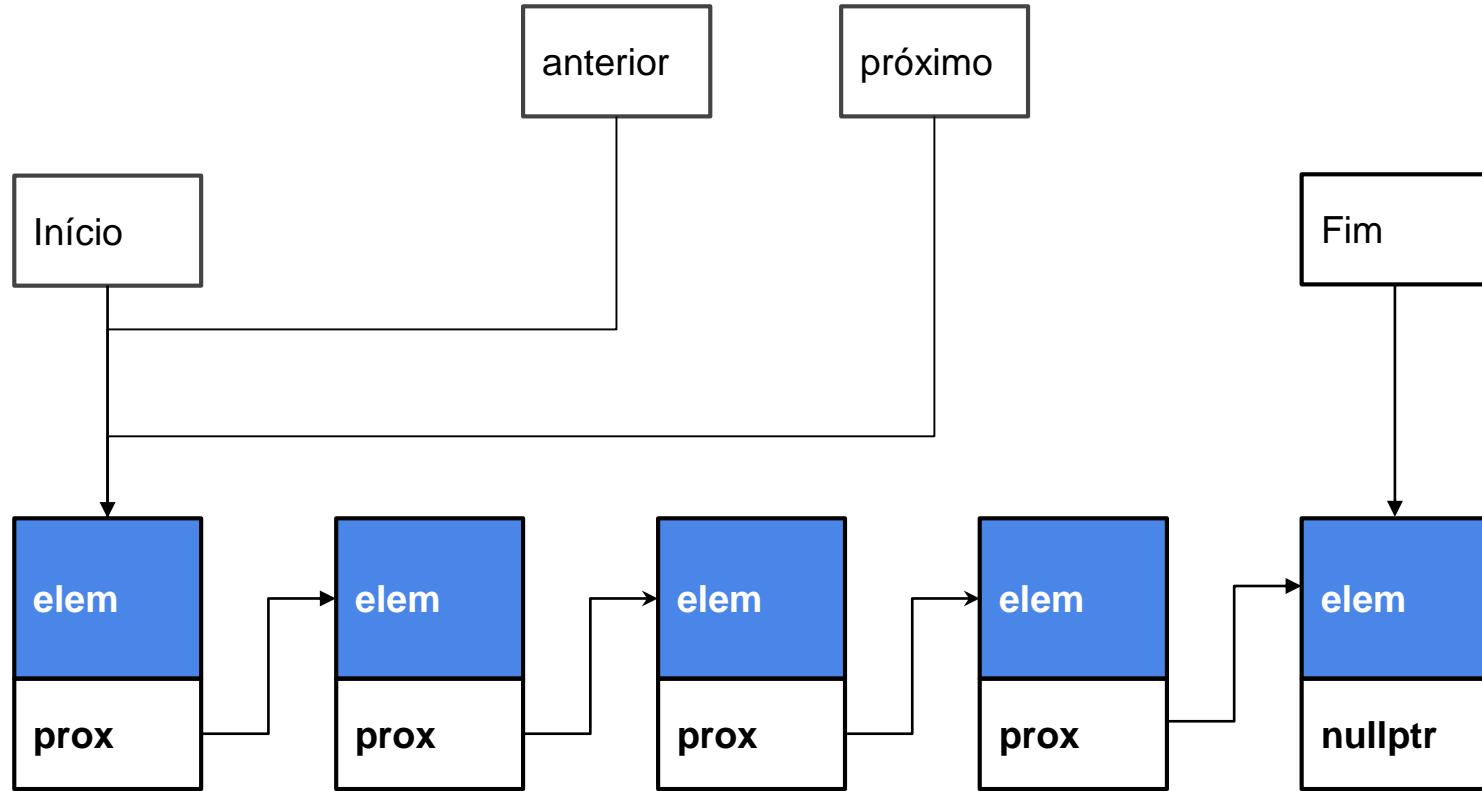
# Qual a ideia?

- Criar um novo elemento
- Setar o valor
- Atualizar o ponteiro do último
- Atualizar o fim da lista!

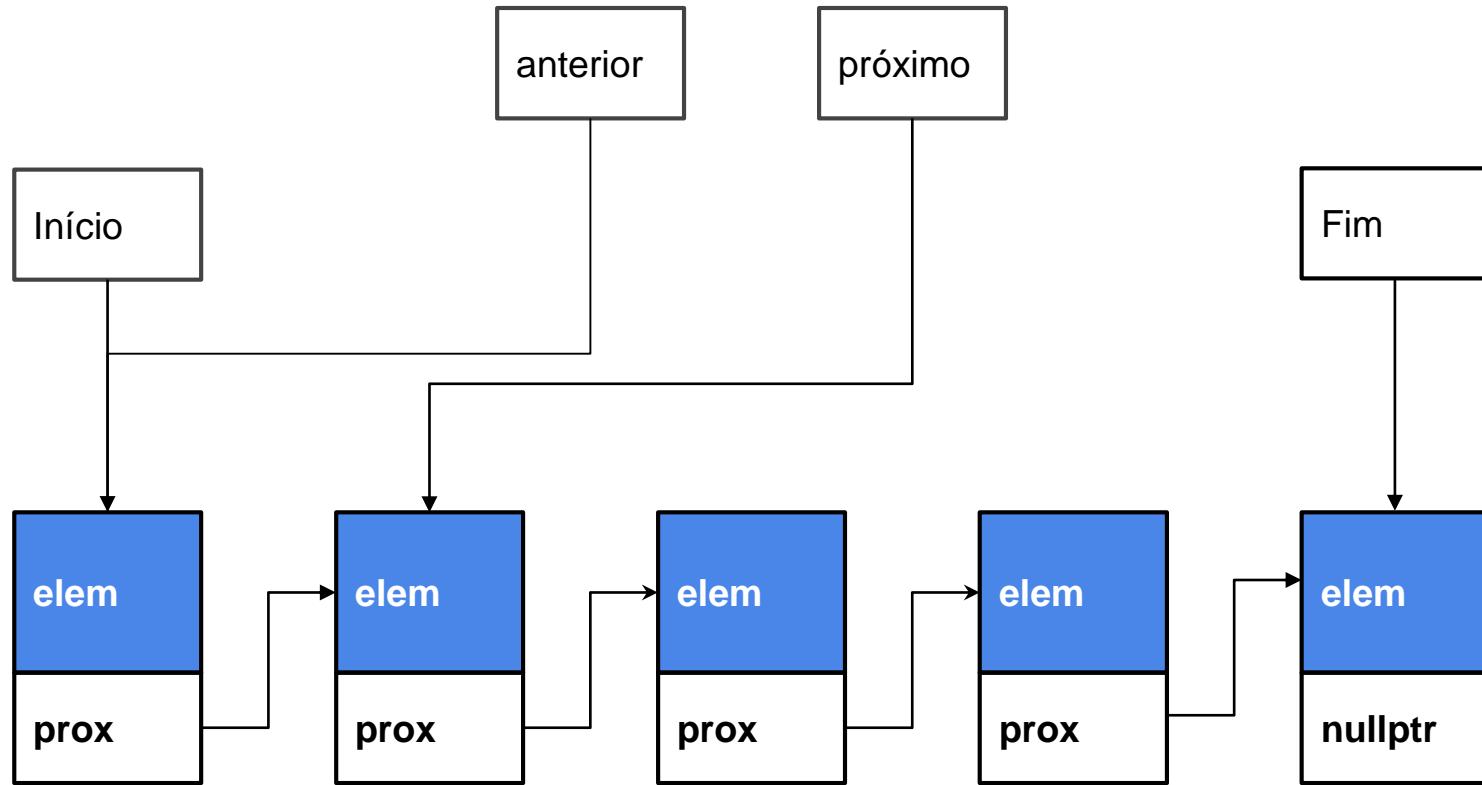
# Destruitor



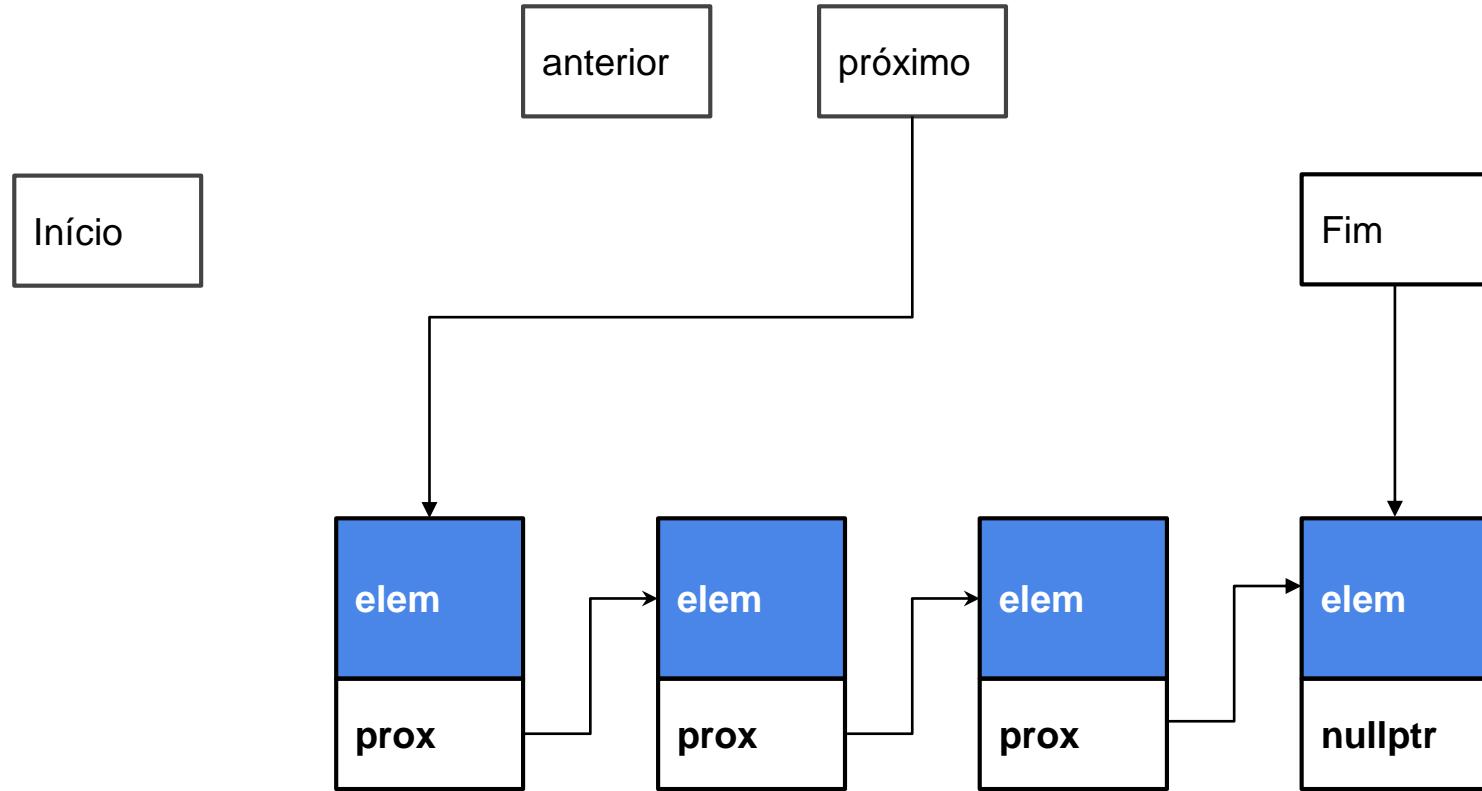
# Destruitor



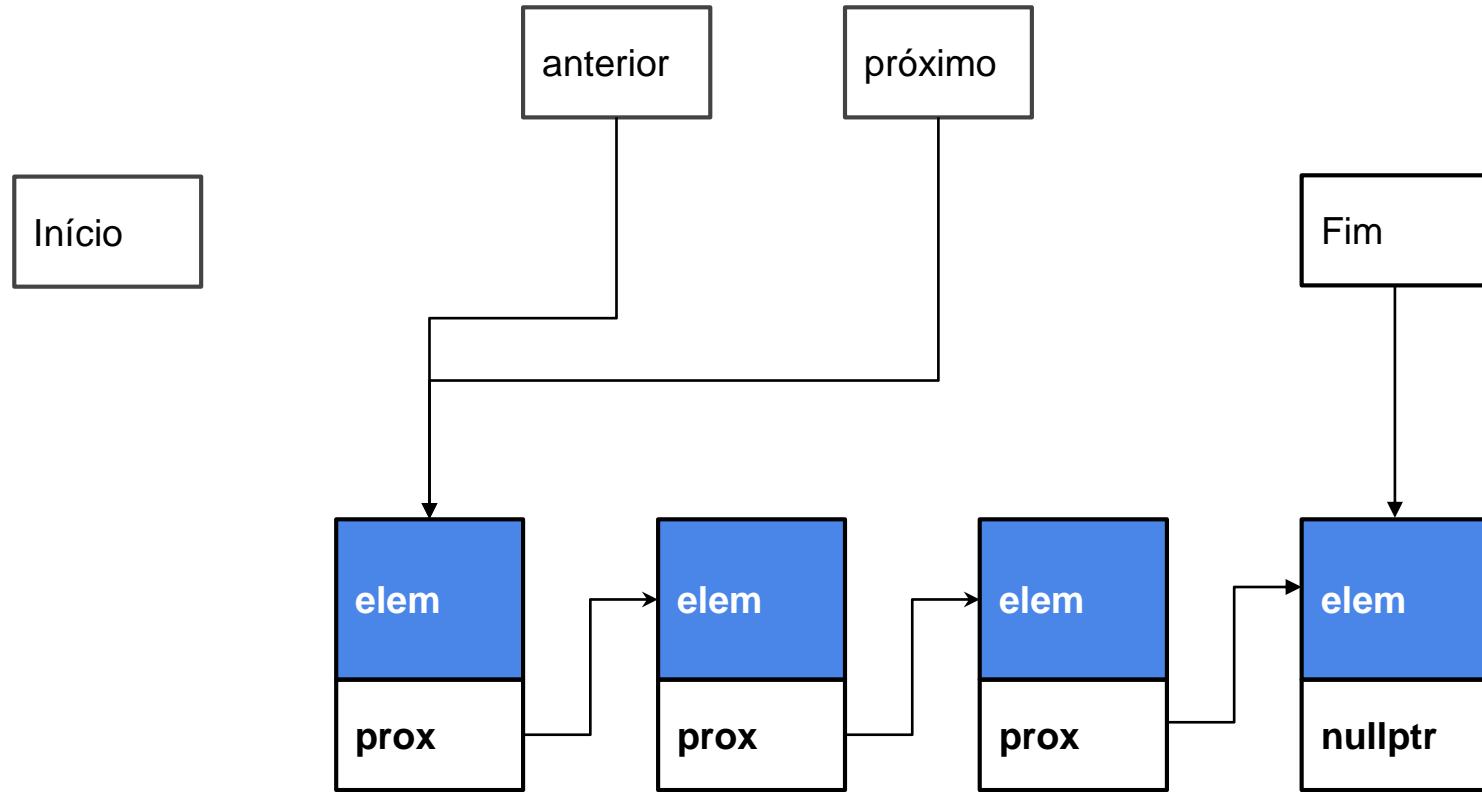
# Destruitor



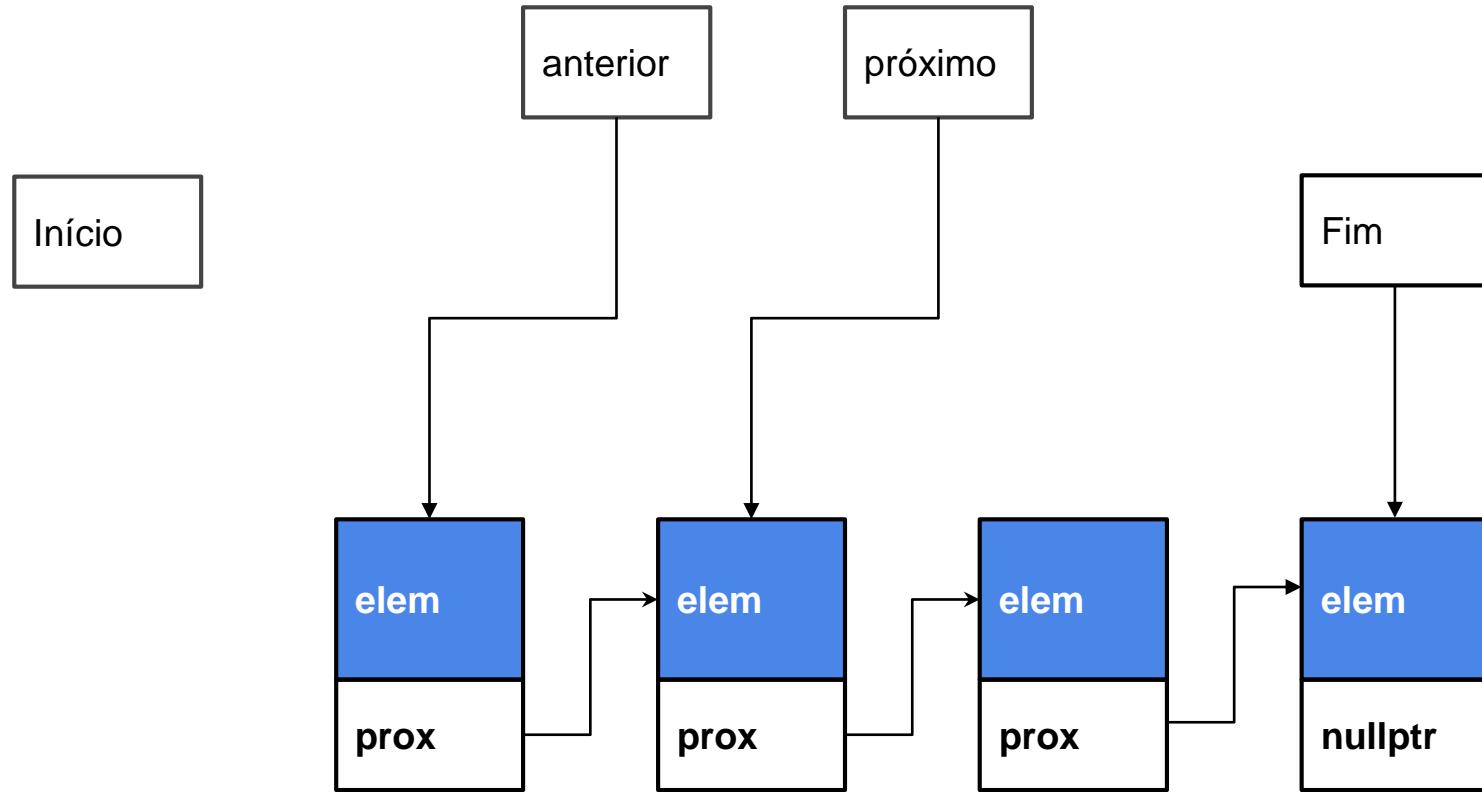
# Destruitor



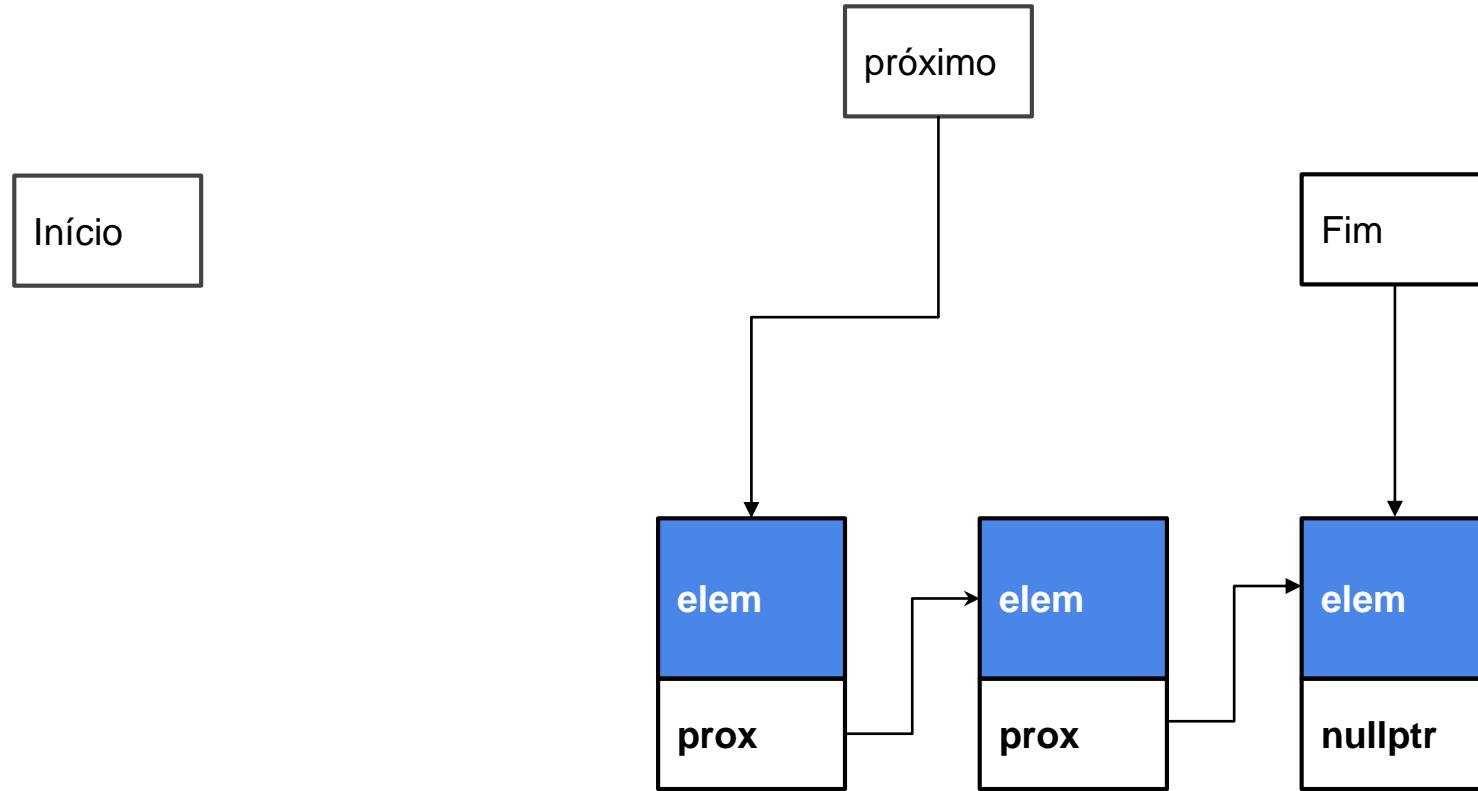
# Destruitor



# Destruitor

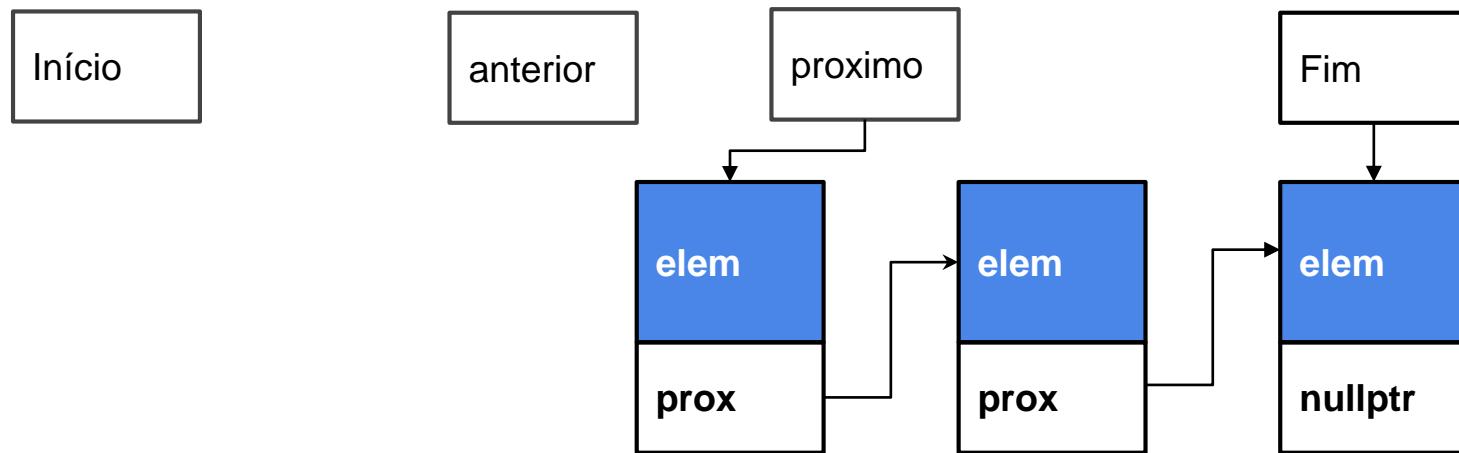


# Destruitor



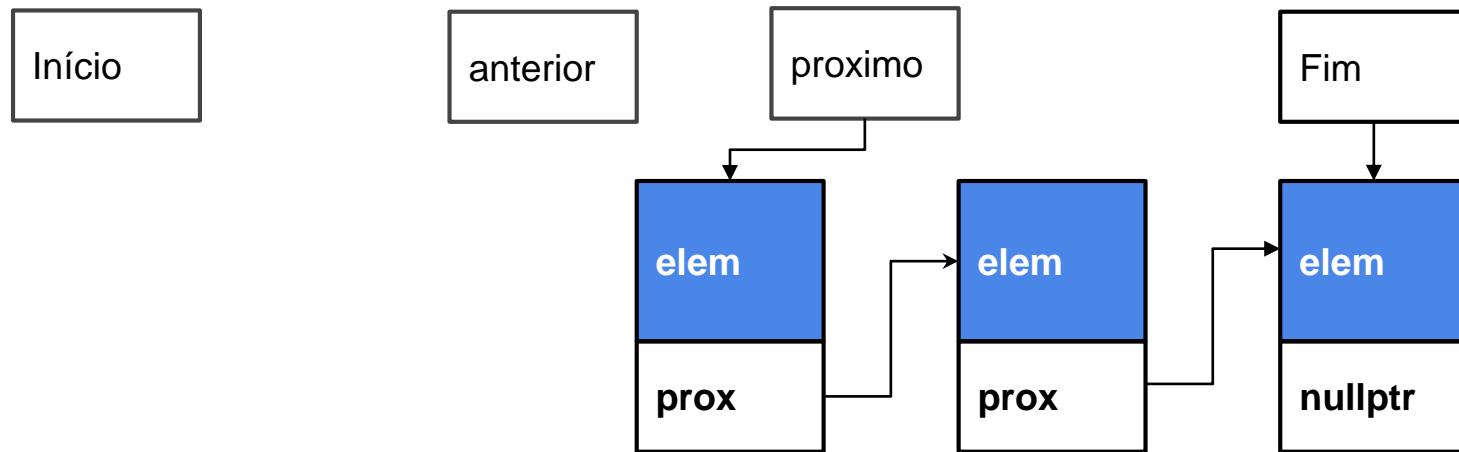
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



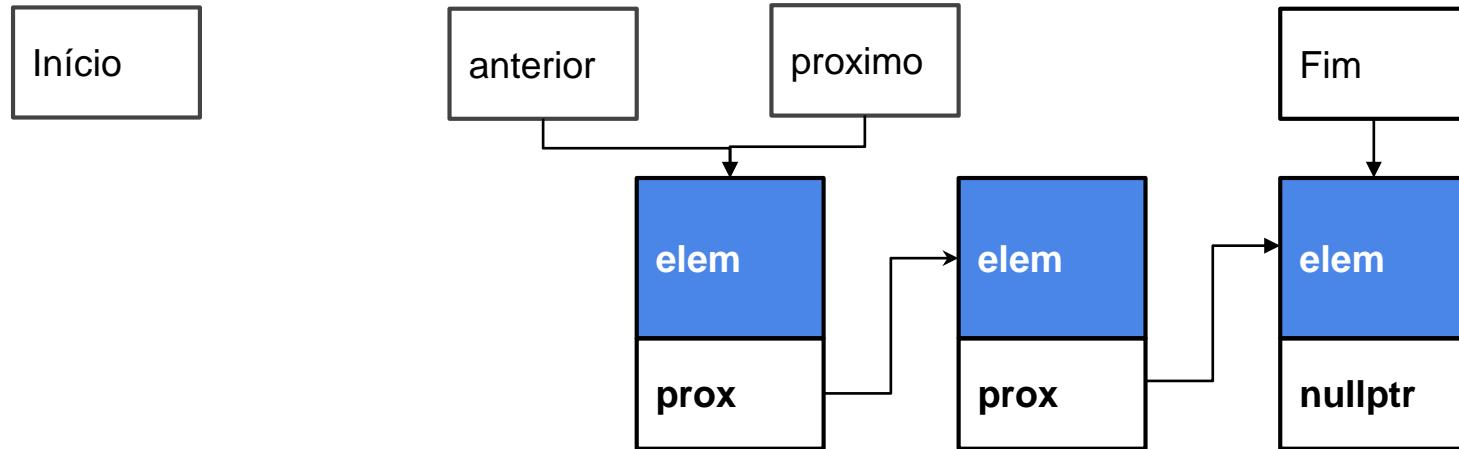
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



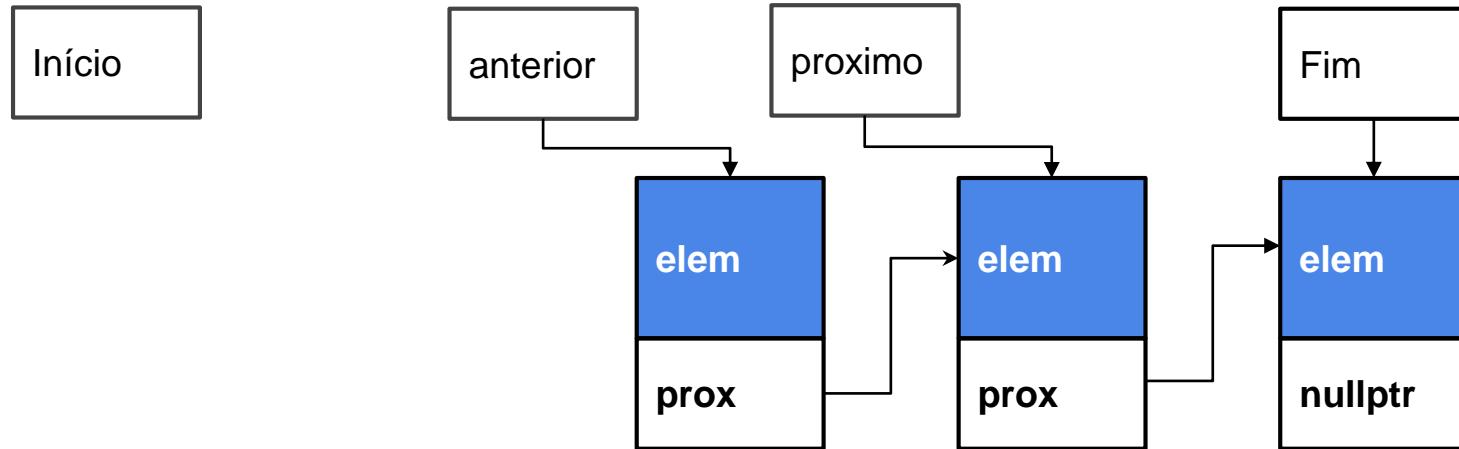
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



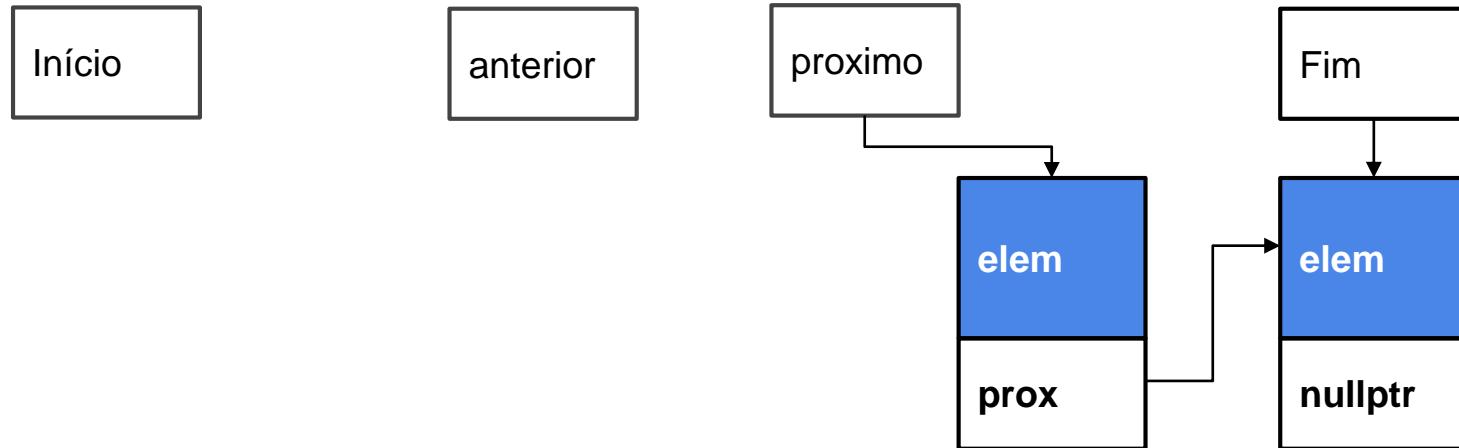
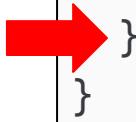
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



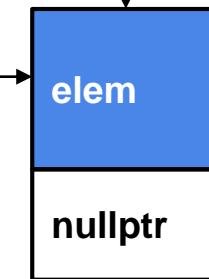
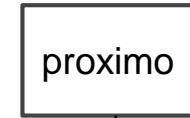
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



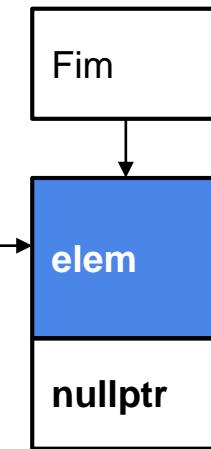
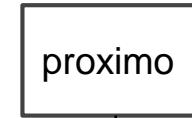
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



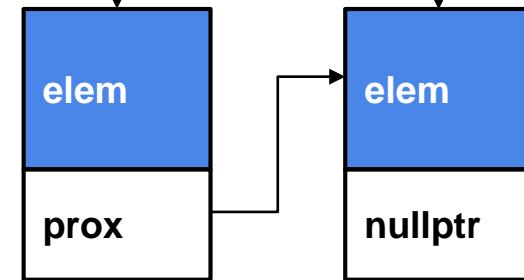
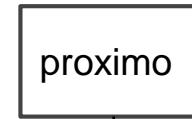
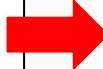
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



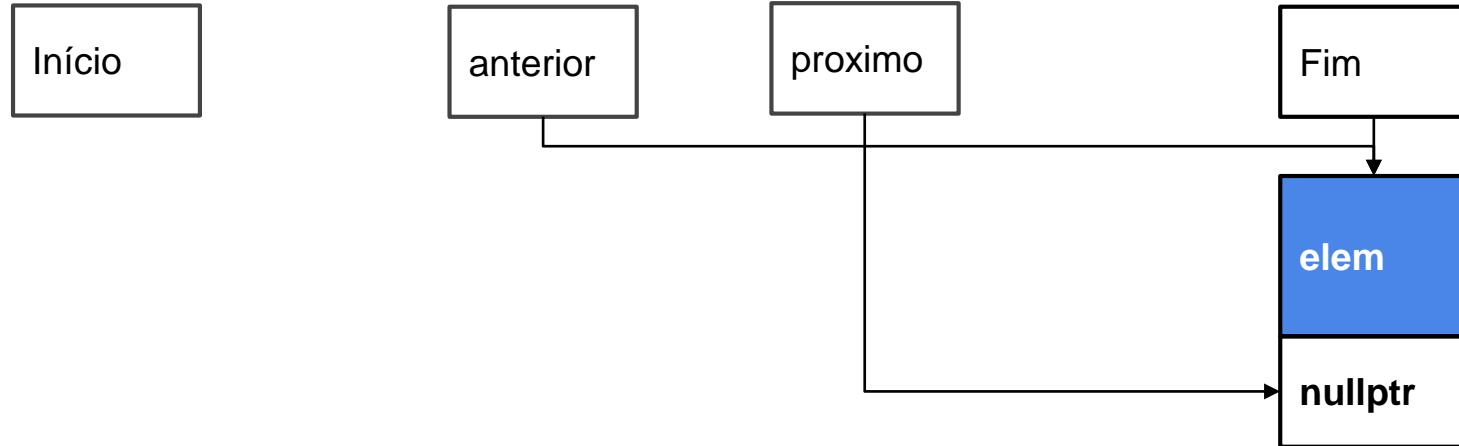
# Destruitor

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



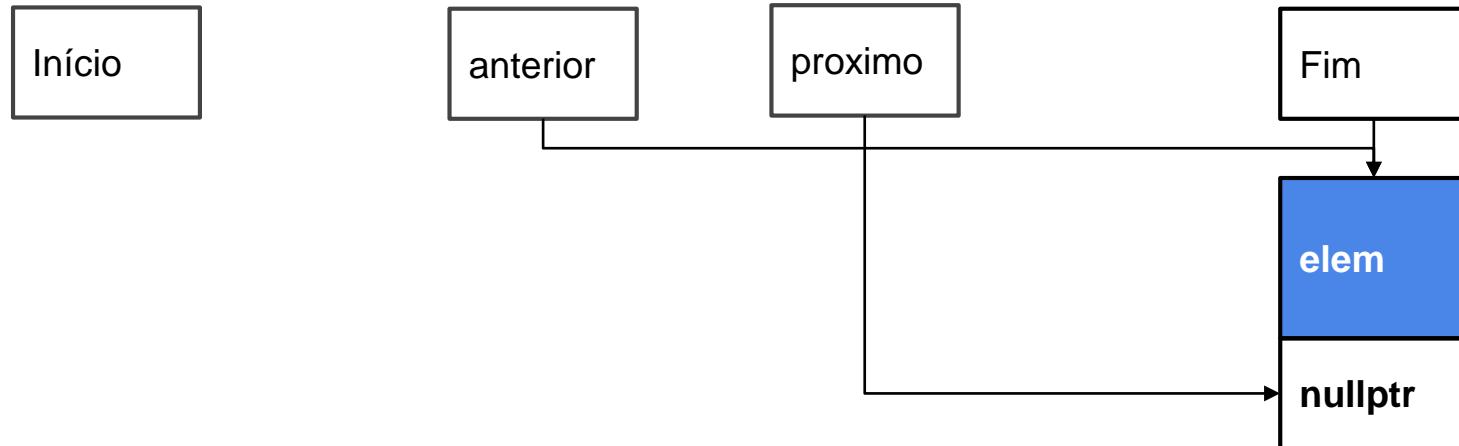
# Neste momento próximo é null

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



# Vai terminar o laço

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



# Vai terminar o laço

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Início

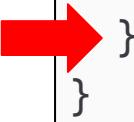
anterior

proximo

Fim

# Fora do laço, ficam os campos sem new

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



Início

anterior

proximo

Fim

# Morrem com o objeto.

```
ListaSimplesmenteEncadeada::~ListaSimplesmenteEncadeada() {  
    node_t *anterior = nullptr;  
    node_t *proximo = _inicio;  
    while (proximo != nullptr) {  
        anterior = proximo;  
        proximo = proximo->proximo;  
        delete anterior;  
    }  
}
```



# **Remoção em uma lista simples**

---

# Código

```
void ListaSimplesmenteEncadeada::remove_iesimo(int i) {
    if (i >= _num_elementos_inseridos)
        return;
    node_t *atual = _inicio;
    node_t *anterior = nullptr;
    for (int j = 0; j < i; j++) {
        anterior = atual;
        atual = atual->proximo;
    }
    if (anterior != nullptr)
        anterior->proximo = atual->proximo;
    if (i == 0)
        _inicio = atual->proximo;
    if (i == _num_elementos_inseridos - 1)
        _fim = anterior;
    _num_elementos_inseridos--;
    delete atual;
}
```

# Código

```
void ListaSimplesmenteEncadeada::remove_iesimo(int i) {
    if (i >= _num_elementos_inseridos)
        return;
    node_t *atual = _inicio;
    node_t *anterior = nullptr;
    for (int j = 0; j < i; j++) {
        anterior = atual;
        atual = atual->proximo;
    }
    if (anterior != nullptr)
        anterior->proximo = atual->proximo;
    if (i == 0)
        _inicio = atual->proximo;
    if (i == _num_elementos_inseridos - 1)
        _fim = anterior;
    _num_elementos_inseridos--;
    delete atual;
}
```

Verifica se "i" é válido

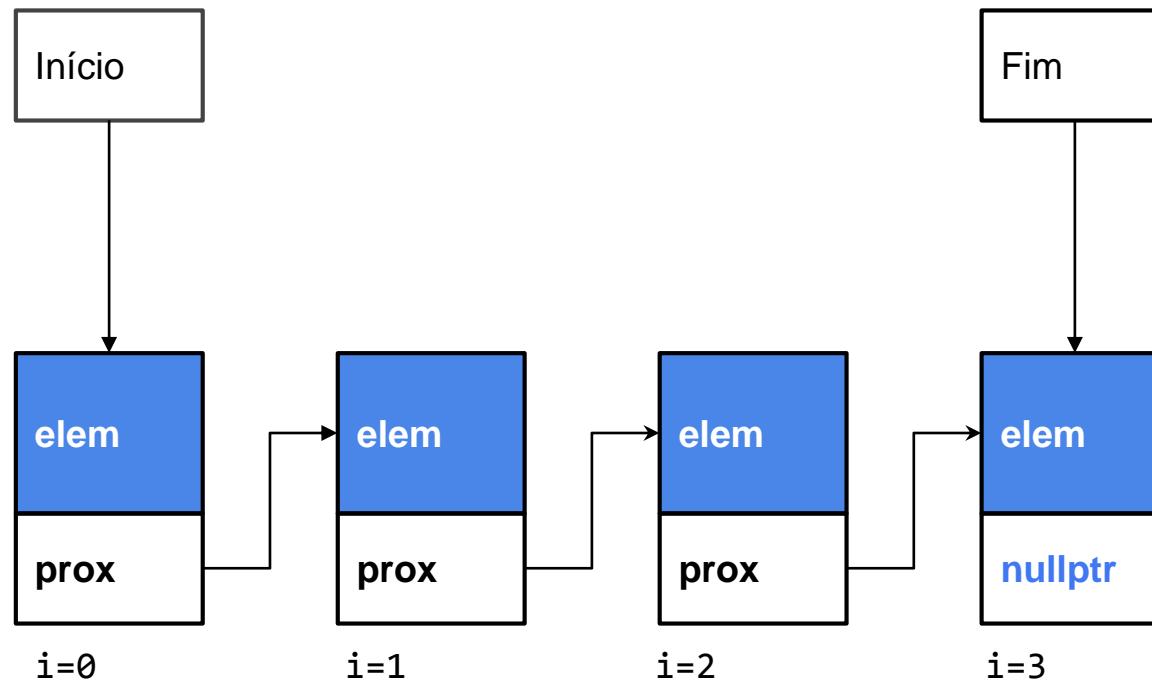
Pula "i" vezes guardando o anterior

Vários "ifs" de casos especiais

# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

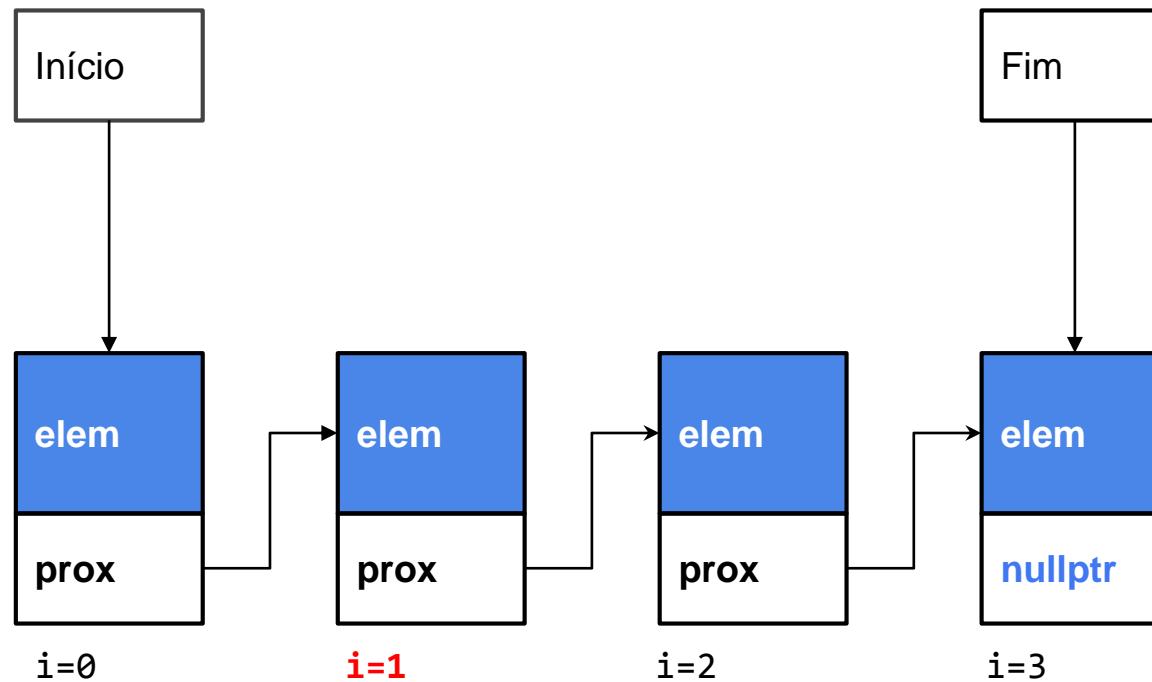
## Olhando para o caso geral



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

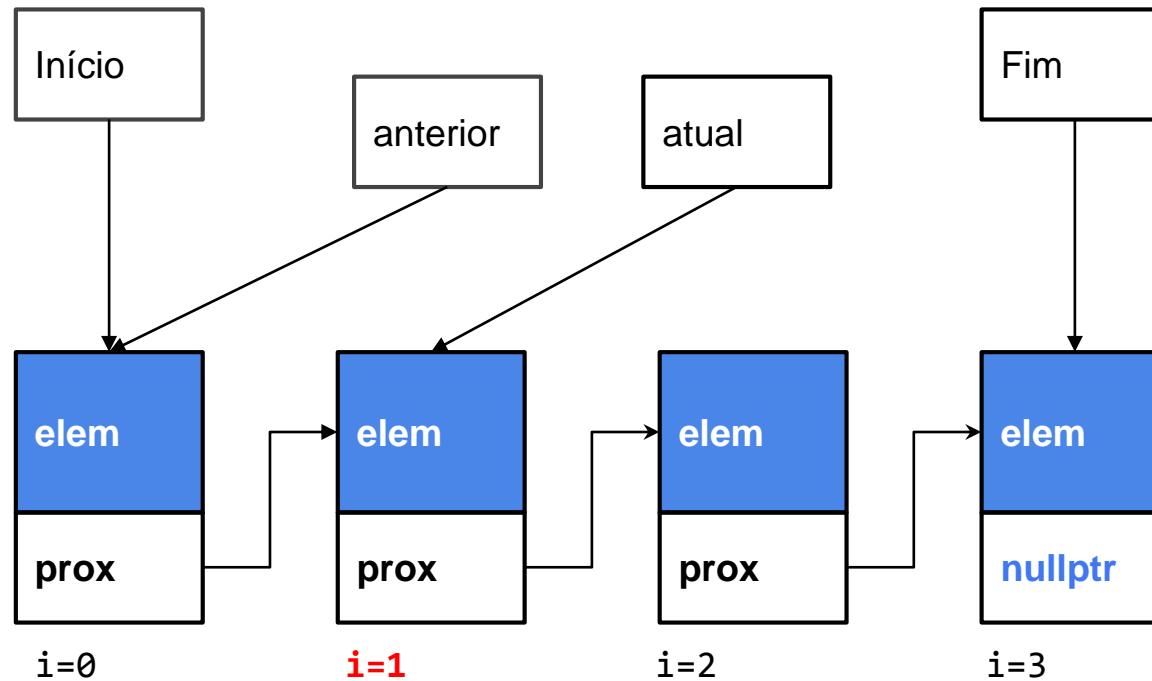
## Olhando para o caso geral ( $i=1$ )



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

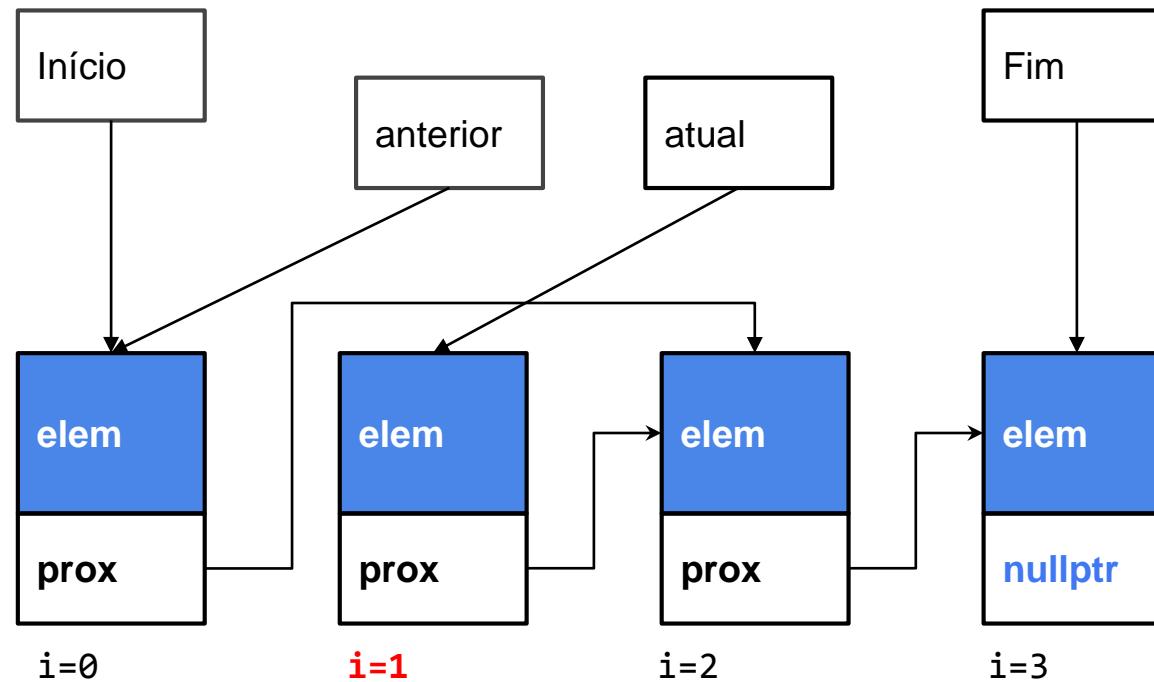
## Caminhinhos guardando o anterior



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

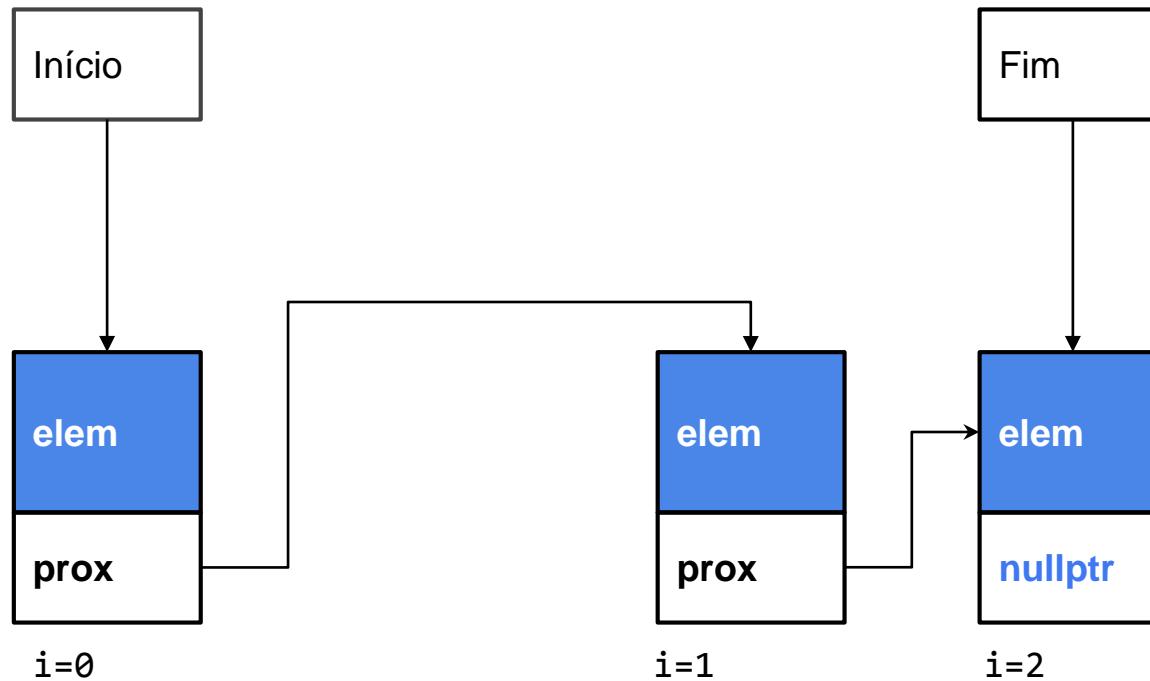
## Atualizamos o ponteiro do anterior



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

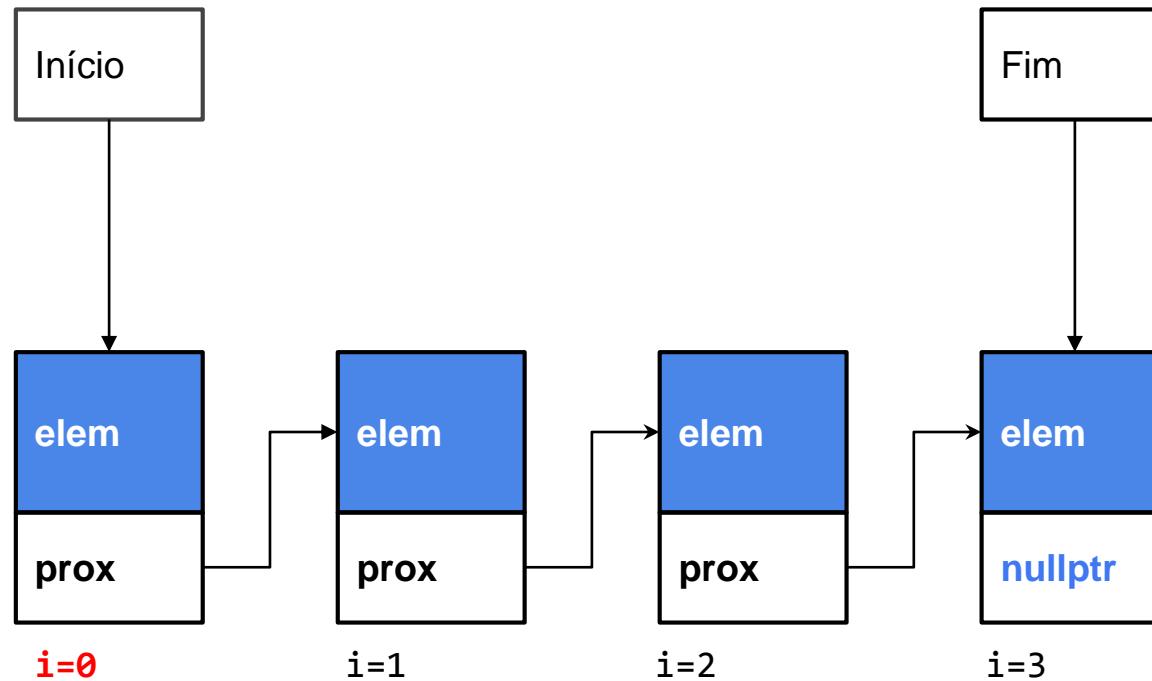
Free!



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

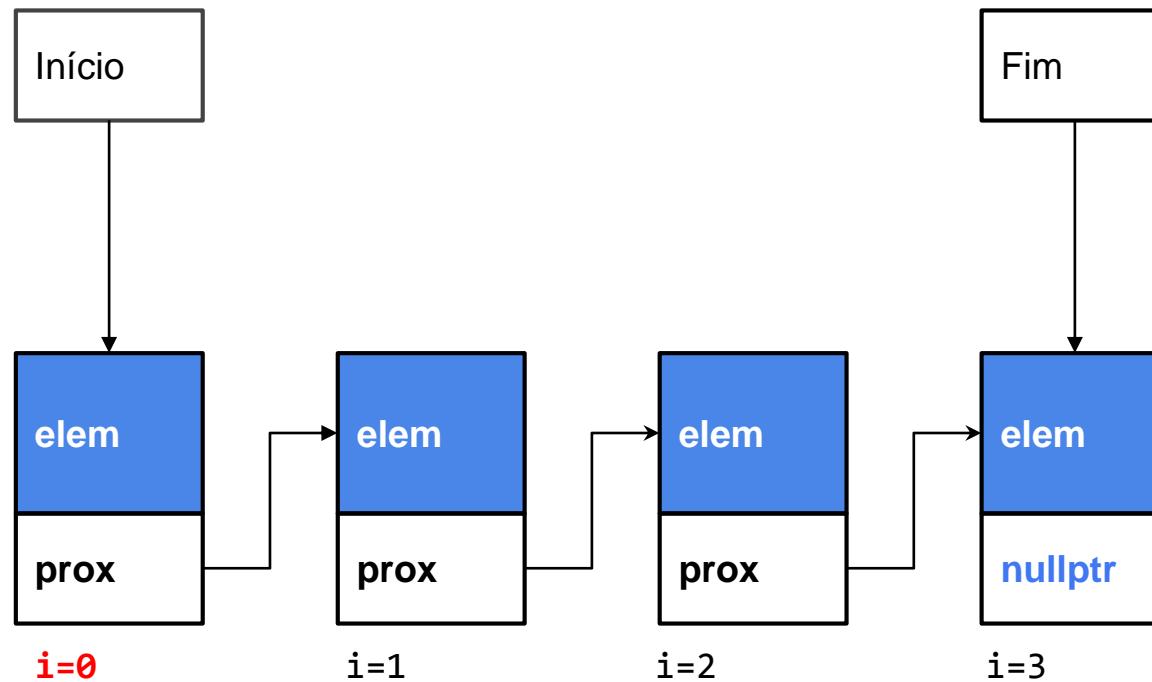
## Caso especial ( $i=0$ )



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

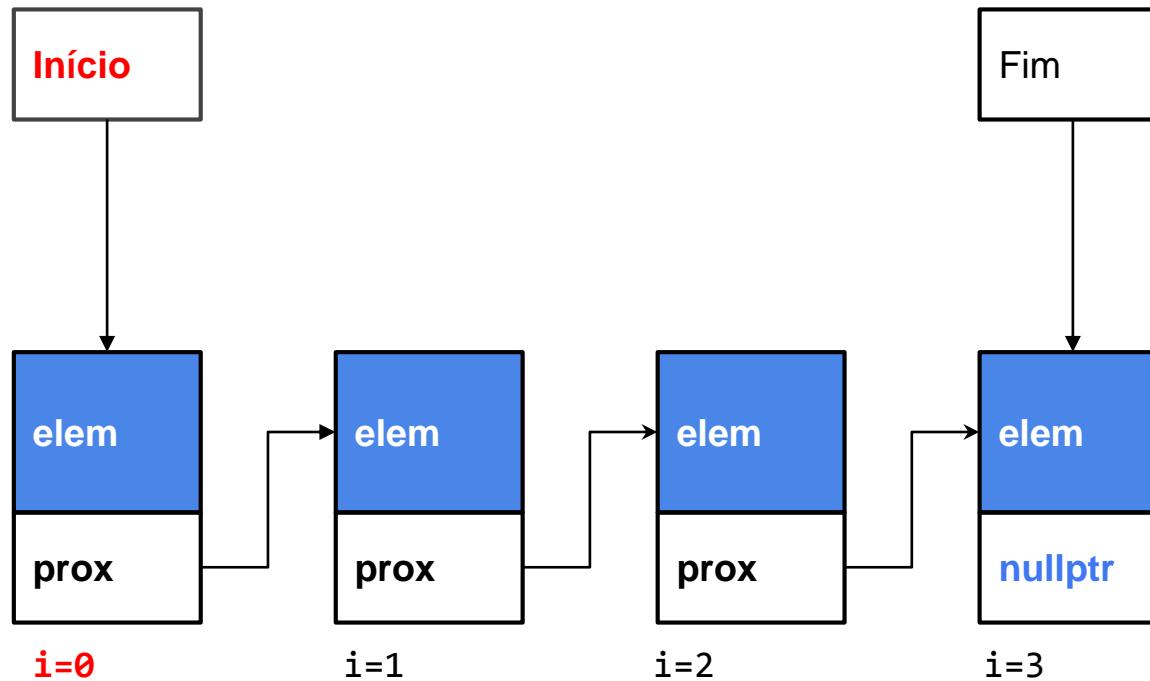
## Qual o problema?



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

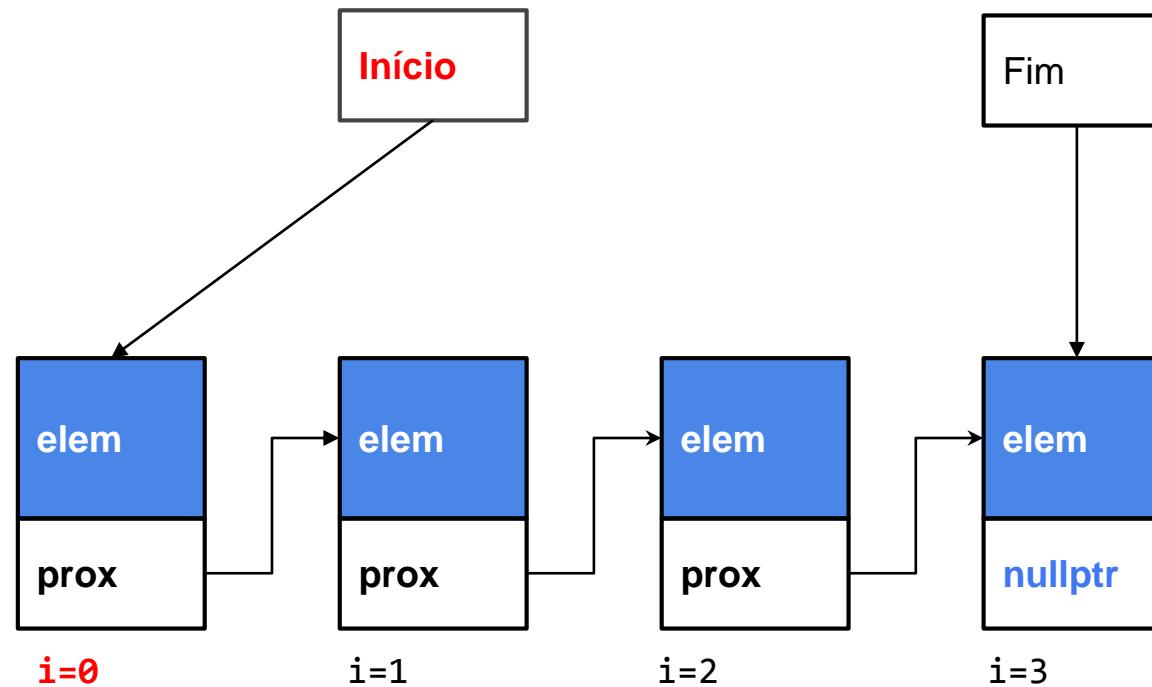
Qual o problema? Ponteiro **início**!



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

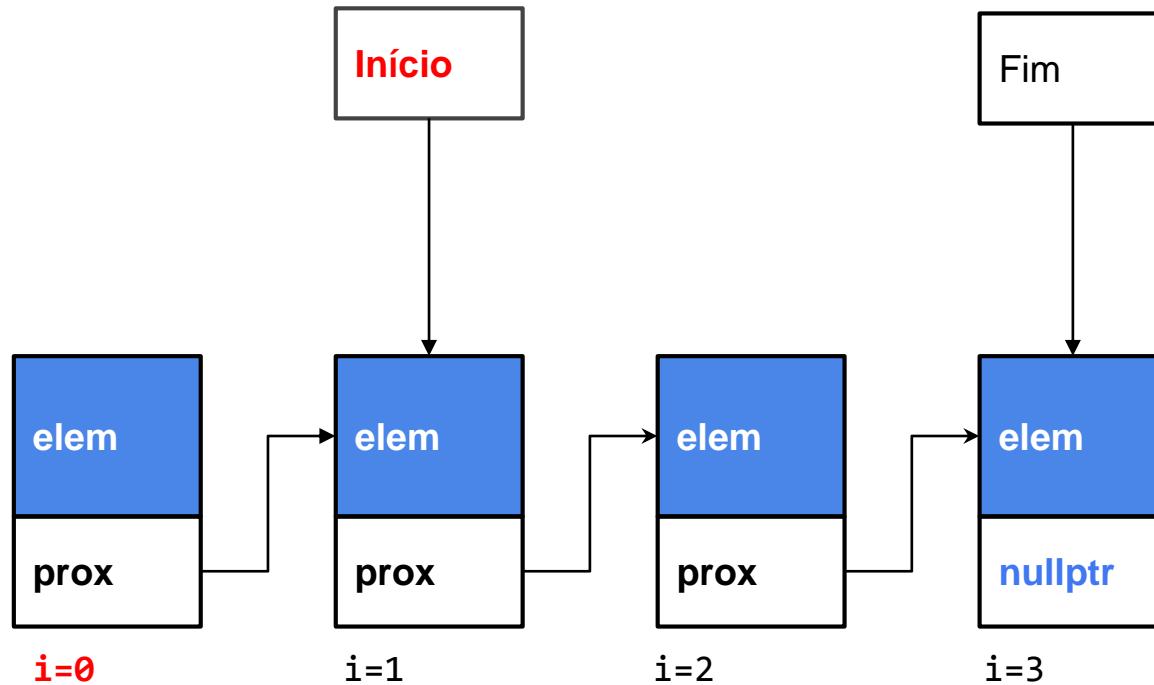
```
if (i == 0)
    _inicio = atual->proxim;
```



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

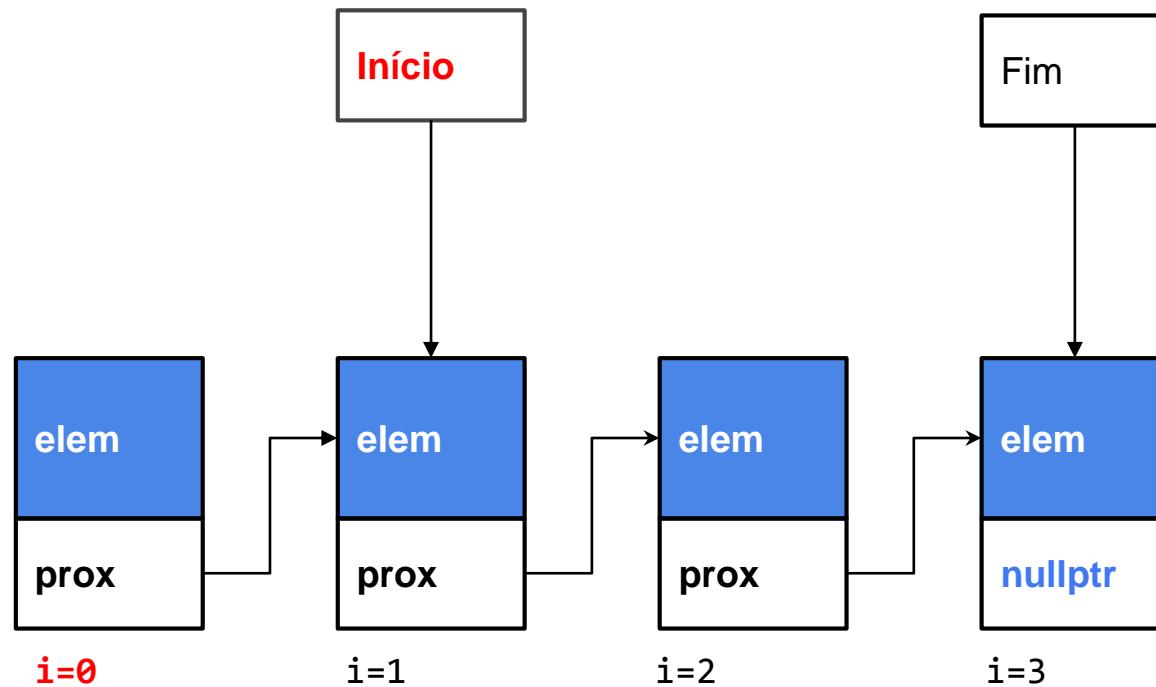
## Mais problemas?



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

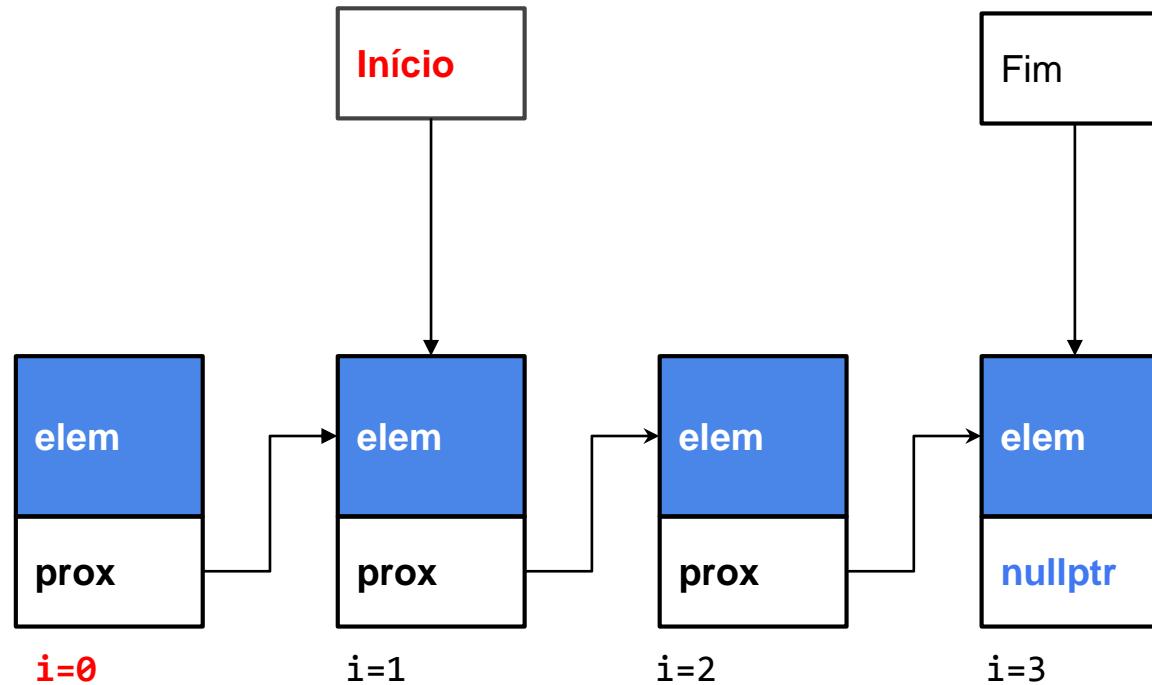
Mais problemas? Valor de anterior é `nullptr`!



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

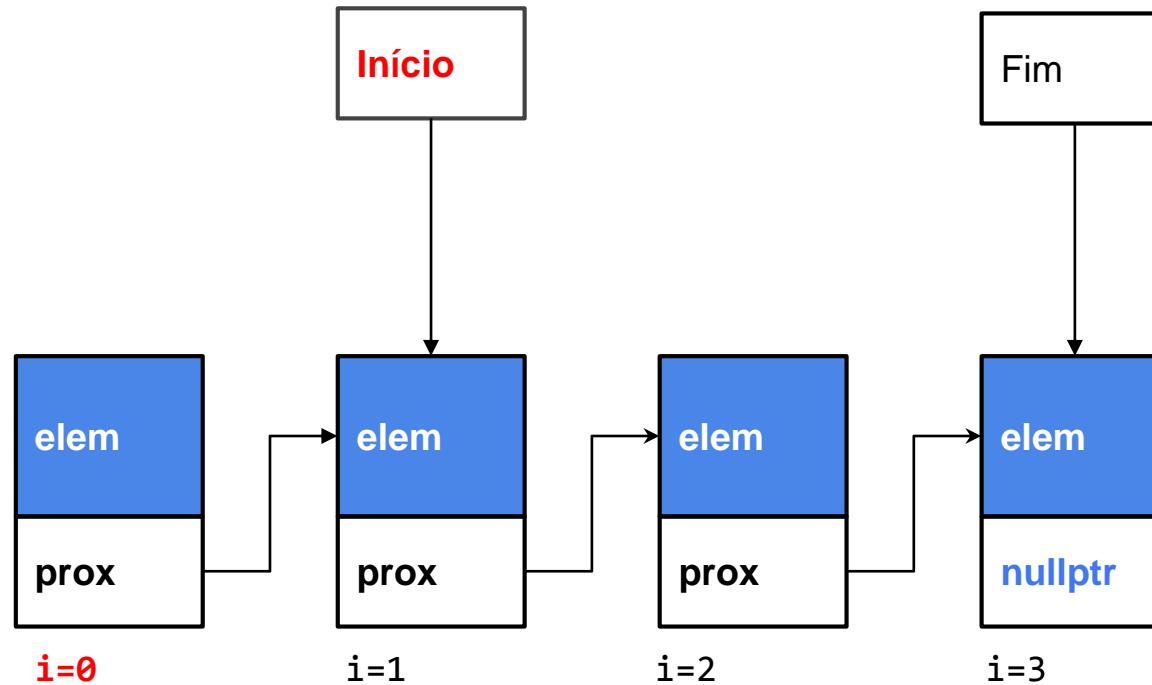
```
if (anterior != nullptr)
    anterior->proximo = atual->proximo;
```



# Casos especiais

Seu código sempre vai ter tratamento para *corner cases*

Sem isto? **segfault!**



# Segfault

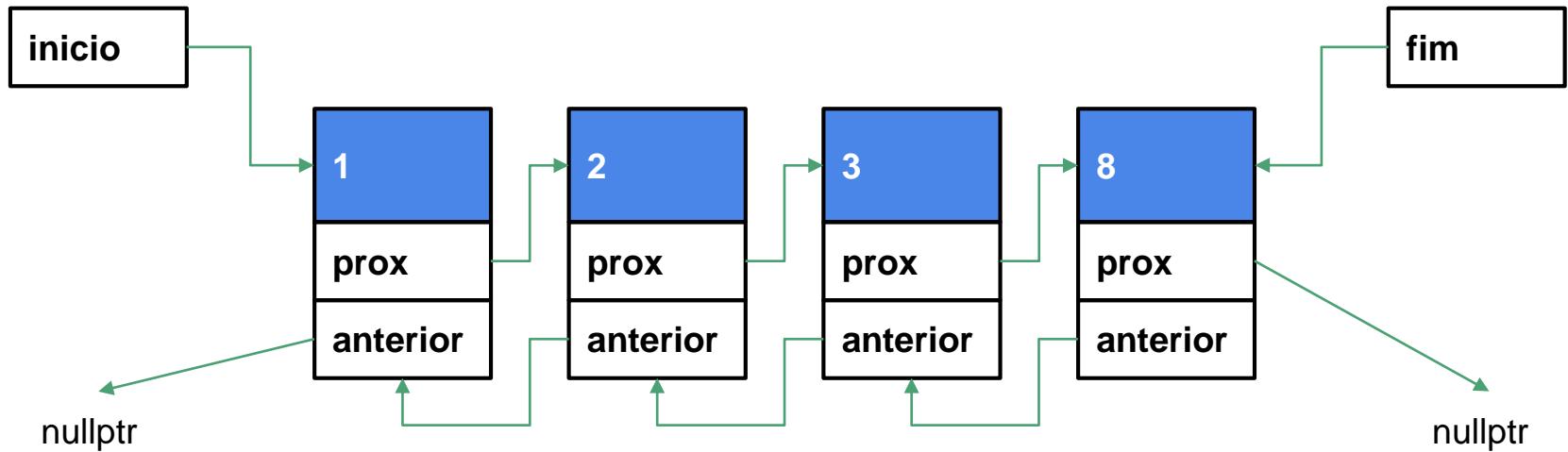
- Ao fazer uso de ponteiros é comum ver segmentation faults
- Causas:
  - Você acessou um ponteiro nulo
  - Você acessou um ponteiro lixo!
- No exemplo anterior existem mais casos especiais
  - Quais são?

# **Lista Duplamente Encadeada**

---

# Lista Duplamente Encadeada

## Ideia



# Olhando para o .h

```
#ifndef PDS2_LISTADUPLA_H
#define PDS2_LISTADUPLA_H
struct node_t {
    int elemento;
    node_t *anterior;
    node_t *proximo;
};

struct ListaDuplamenteEncadeada {
    node_t *_inicio;
    node_t *_fim;
    int _num_elementos_inseridos;
    :
    ListaDuplamenteEncadeada();
    ~ListaDuplamenteEncadeada();
    void inserir_elemento(int elemento);
    void remove_iesimo(int i);
    void imprimir();
};
#endif
```

Ponteiros para o anterior e próximo

Essencialmente a mesma coisa de antes

# Vantagens

Em relação à lista simples...

- Como imprimir nós nas duas ordens
  - Complicado na lista simples
  - Podemos iterar nos dois sentidos
- Não precisamos ficar guardando o ponteiro anterior
- Já existe na estrutura

# Código de remover i-ésimo

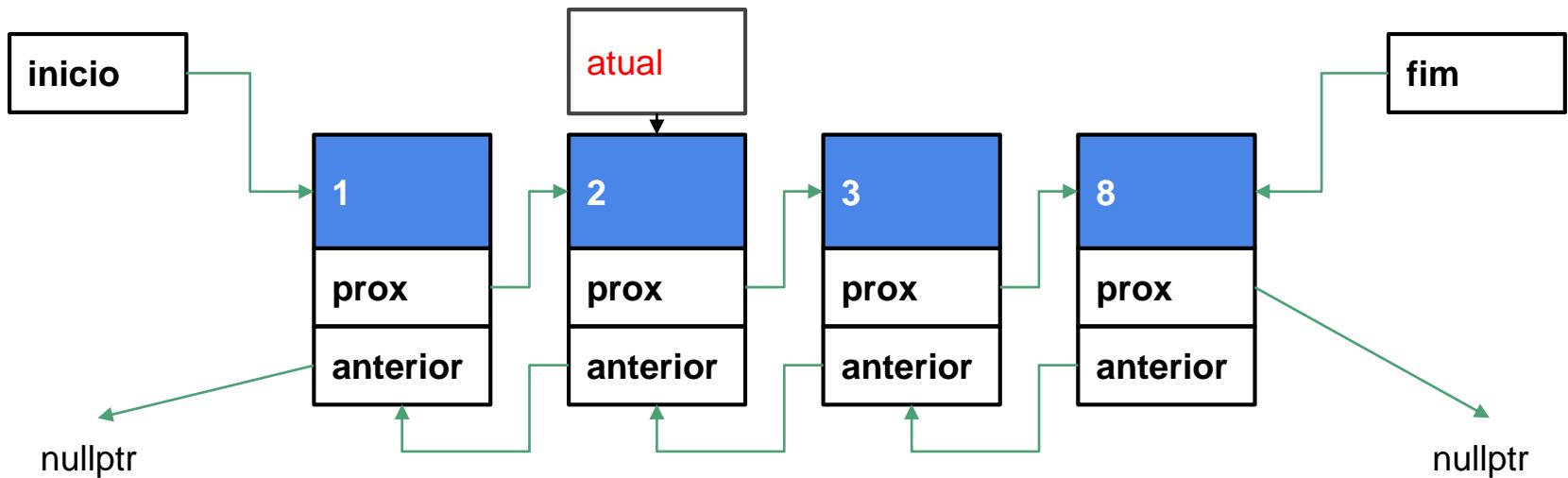
Um pouco mais simples do que antes

```
void ListaDuplamenteEncadeada::remove_iesimo(int i) {
    if (i >= _num_elementos_inseridos) {
        return;
    }
    node_t *atual = _inicio;
    for (int j = 0; j < i; j++)
        atual = atual->proximo;
    if (atual->proximo != nullptr)
        atual->proximo->anterior = atual->anterior;
    if (atual->anterior != nullptr)
        atual->anterior->proximo = atual->proximo;
    if (i == 0)
        _inicio = atual->proximo;
    if (i == _num_elementos_inseridos - 1)
        _fim = atual->anterior;
    _num_elementos_inseridos--;
    delete atual;
}
```

# Código de remover i-ésimo

Um pouco mais simples do que antes

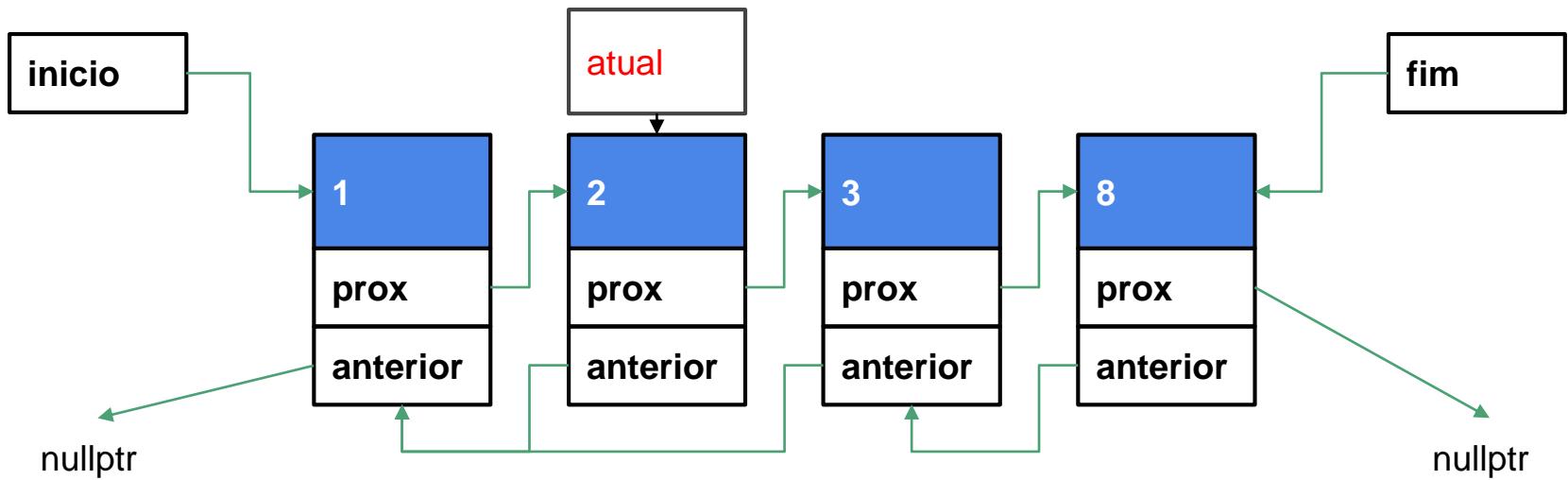
```
void ListaDuplamenteEncadeada::remove_iesimo(int i) {  
    // ...  
    if (atual->proximo != nullptr)  
        atual->proximo->anterior = atual->anterior;  
    if (atual->anterior != nullptr)  
        atual->anterior->proximo = atual->proximo;  
    // ...
```



# Código de remover i-ésimo

Um pouco mais simples do que antes

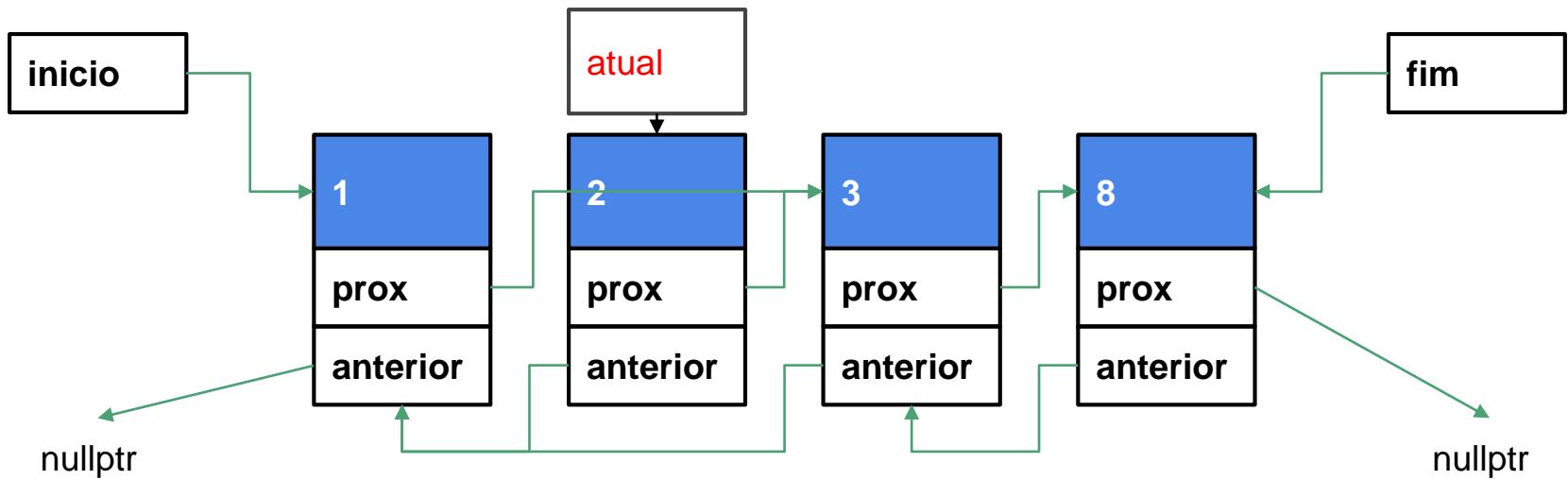
```
void ListaDuplamenteEncadeada::remove_iesimo(int i) {  
    // ...  
    if (atual->proximo != nullptr)  
        atual->proximo->anterior = atual->anterior;  
    if (atual->anterior != nullptr)  
        atual->anterior->proximo = atual->proximo;  
    // ...
```



# Código de remover i-ésimo

Um pouco mais simples do que antes

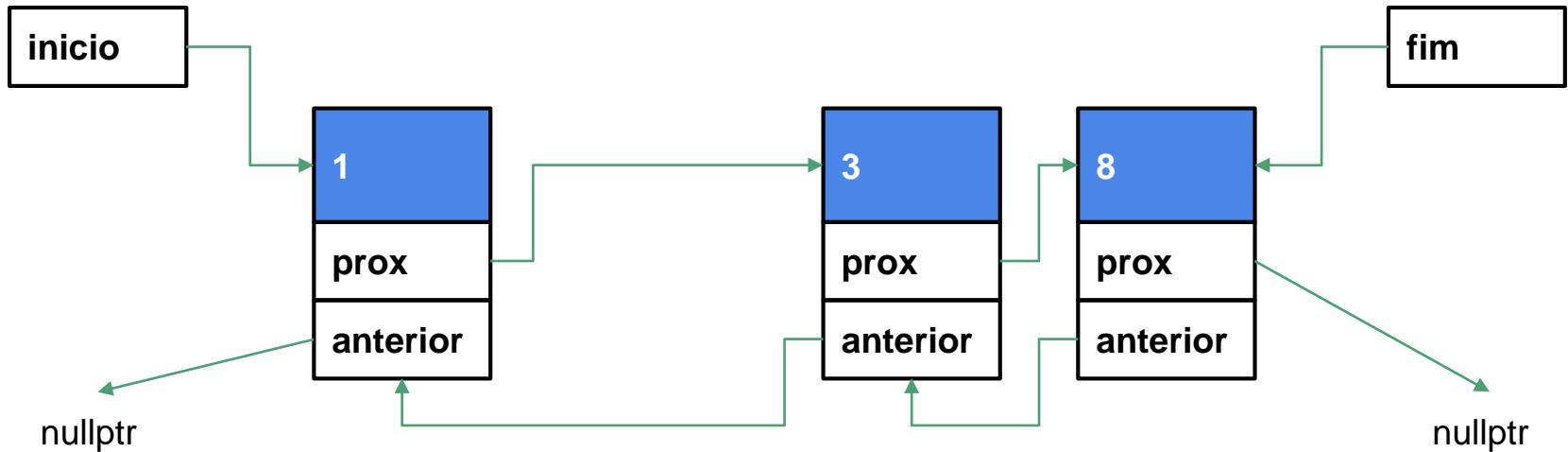
```
void ListaDuplamenteEncadeada::remove_iesimo(int i) {  
    // ...  
    if (atual->proximo != nullptr)  
        atual->proximo->anterior = atual->anterior;  
    if (atual->anterior != nullptr)  
        atual->anterior->proximo = atual->proximo;  
    // ...
```



# Código de remover i-ésimo

Um pouco mais simples do que antes

```
void ListaDuplamenteEncadeada::remove_iesimo(int i) {  
    // ...  
    if (atual->proximo != nullptr)  
        atual->proximo->anterior = atual->anterior;  
    if (atual->anterior != nullptr)  
        atual->anterior->proximo = atual->proximo;  
    // ...
```



# Problemas que precisam de listas

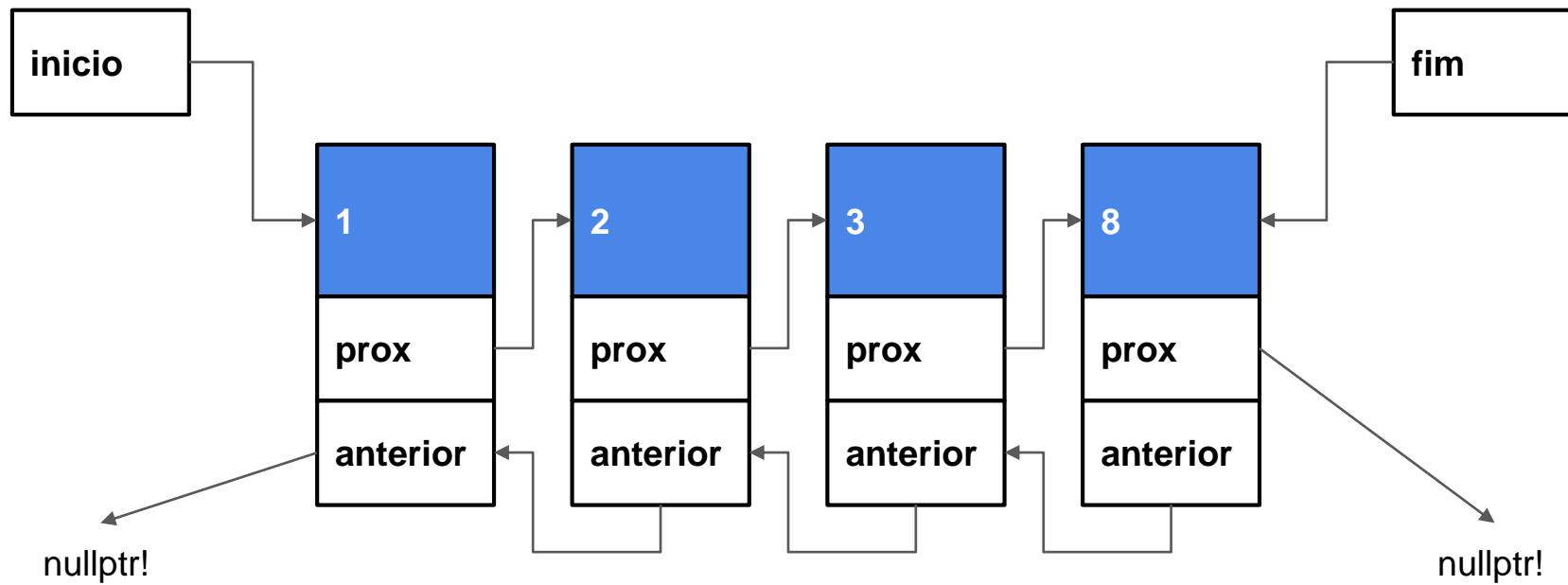
Ou de conceitos similares

- Tipos simples de dados têm limites
  - int, float, double, long
- Como representar números gigantescos?
  - Sequência de dígitos

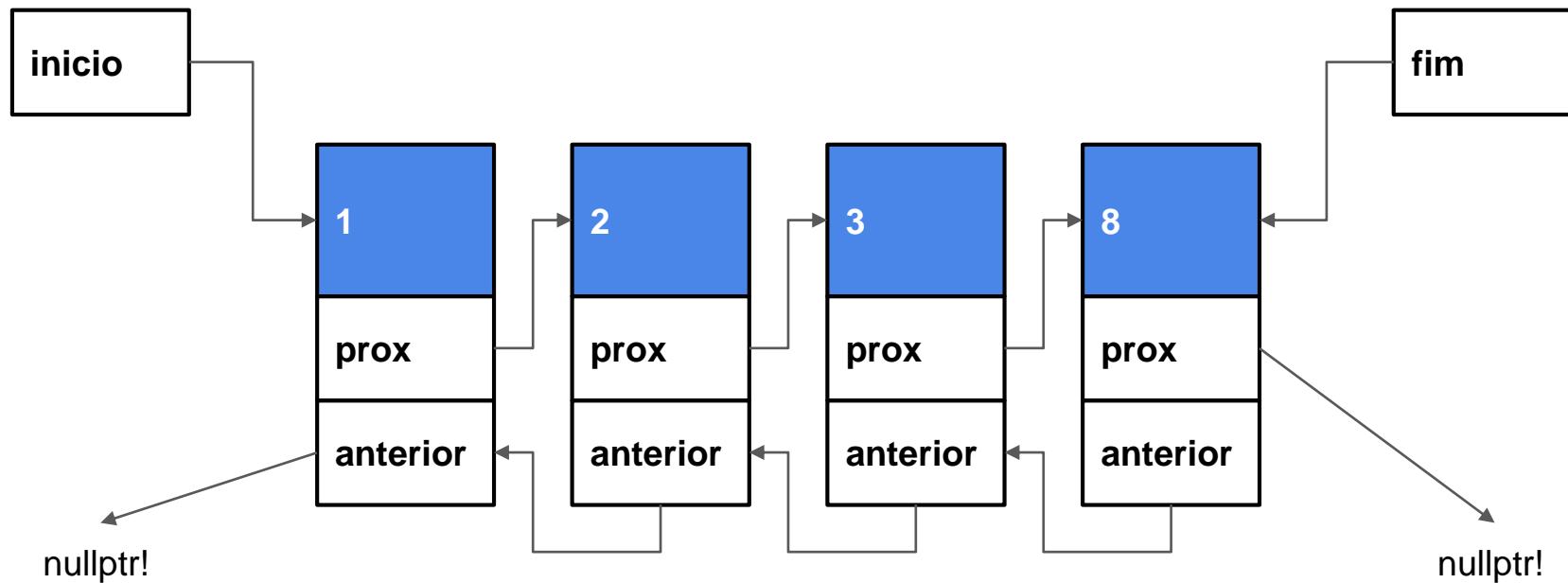
# **TAD Big Num**

---

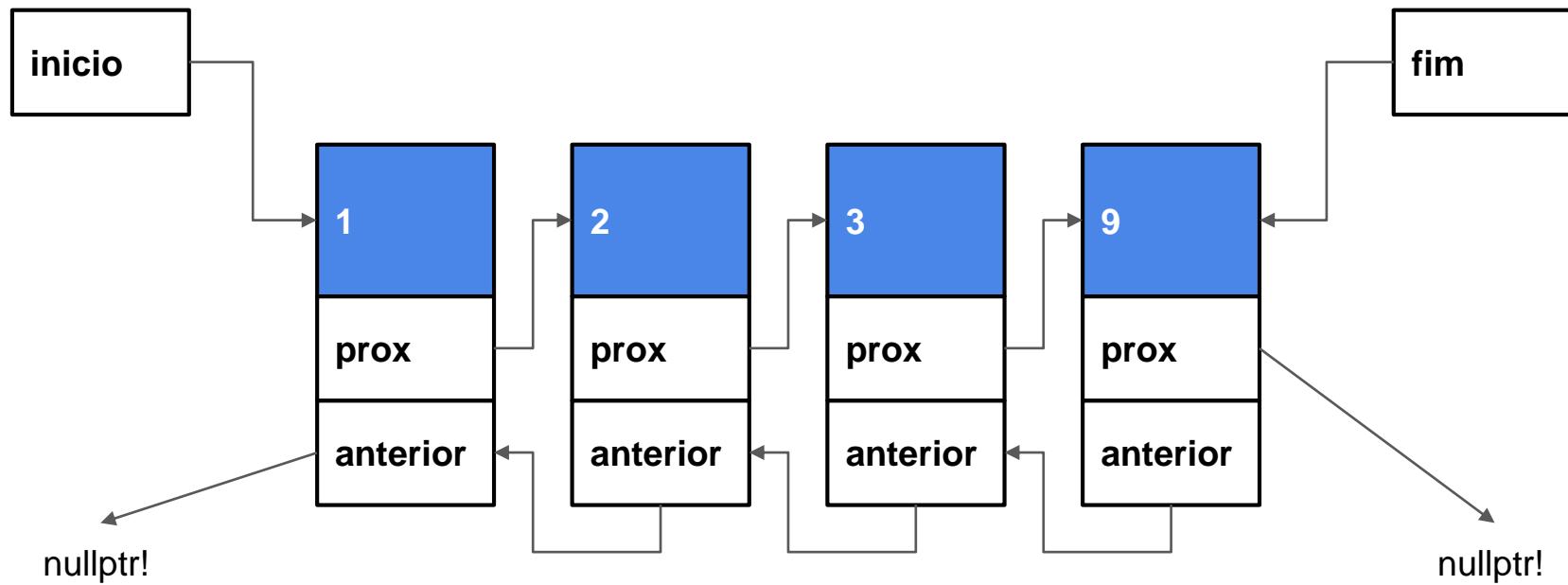
# Lista duplamente encadeada: 1238 abaixo



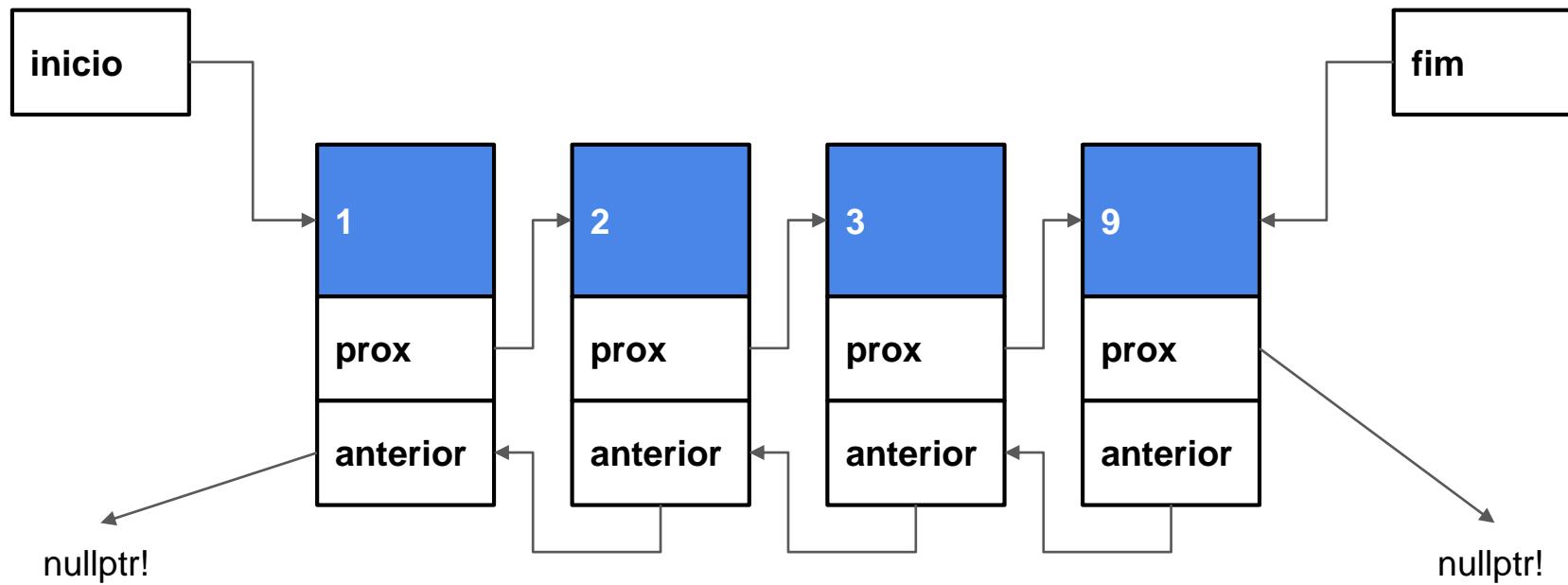
$$1238 + 1 = 1239$$



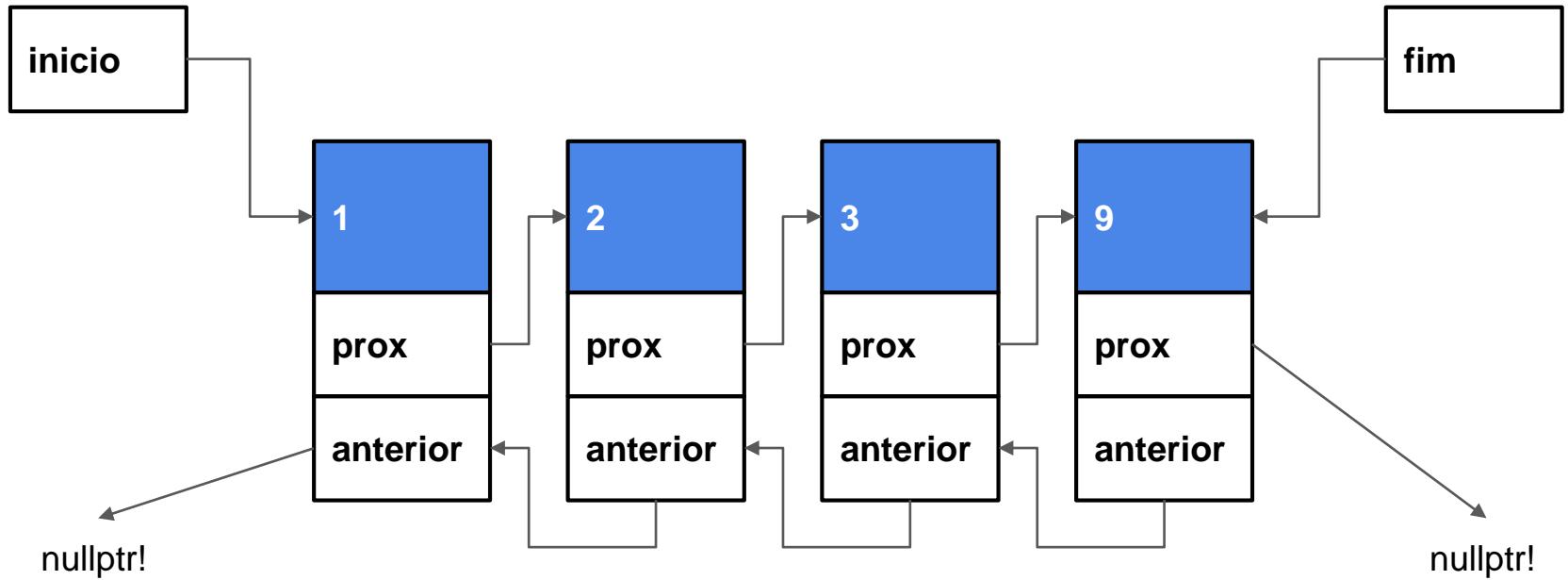
$$1238 + 1 = 1239$$



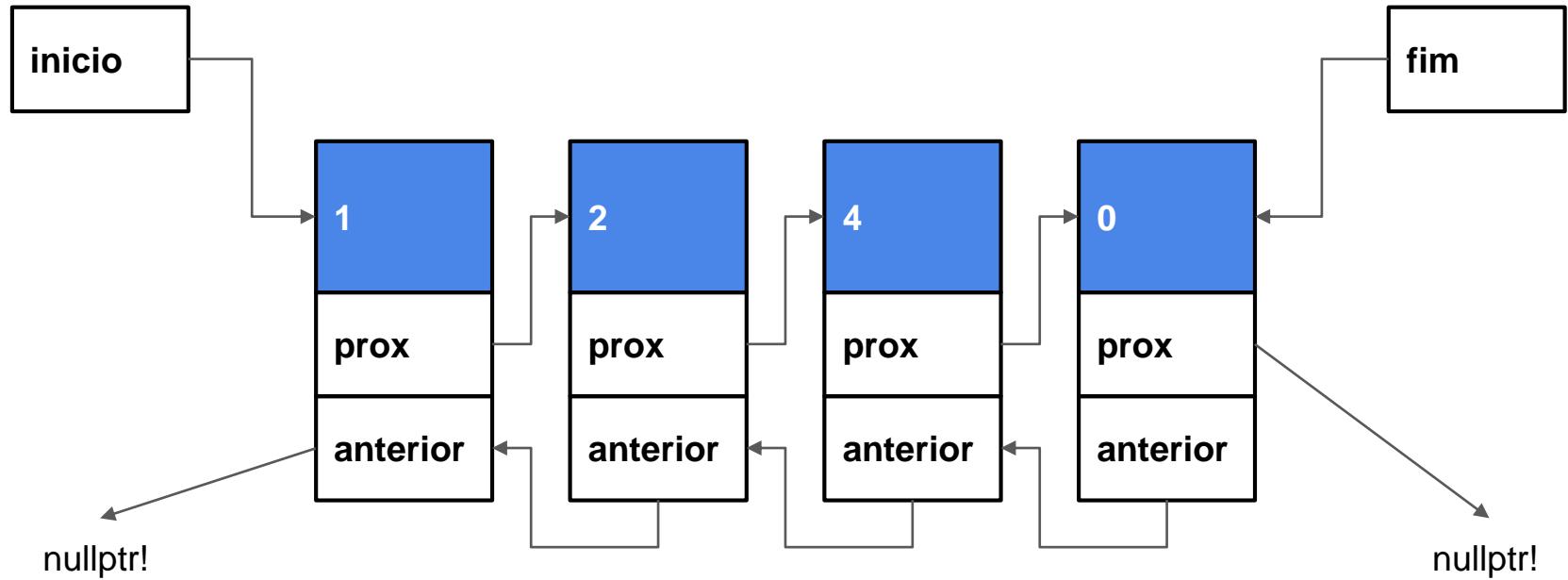
$$1239 + 1 = 1240$$



vai  $\downarrow$  para frente: zeramos o menor



vai  $\downarrow$  para frente: zeramos o menor



- Note que ainda é uma lista
- Mudamos as operações
- Novo TAD
  - Memória similar
  - Operações diferentes
- Já fizemos o oposto
  - Mesmas operações
  - Mesmo TAD

```
#ifndef PDS2_BIGNUM_H
#define PDS2_BIGNUM_H
struct node_t {
    int valor;
    node_t *anterior;
    node_t *proximo;
};

struct BigNum {

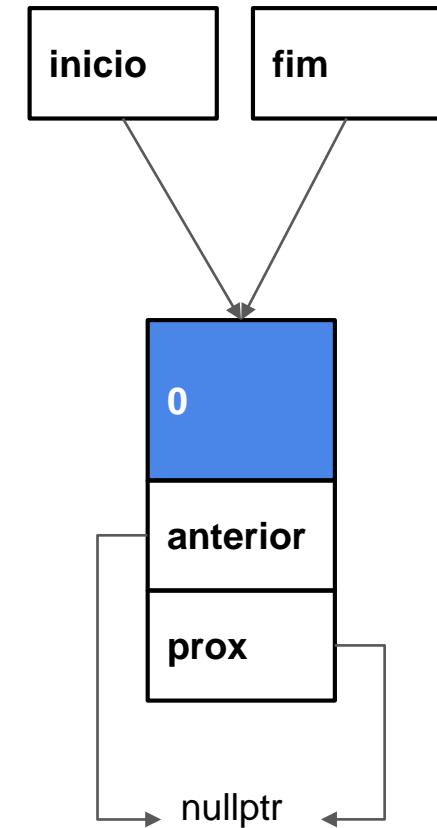
    node_t *_inicio;
    node_t *_fim;

    BigNum();
    ~BigNum();

    void incrementa();
    void decrementa();
    void imprimir();
};
#endif
```

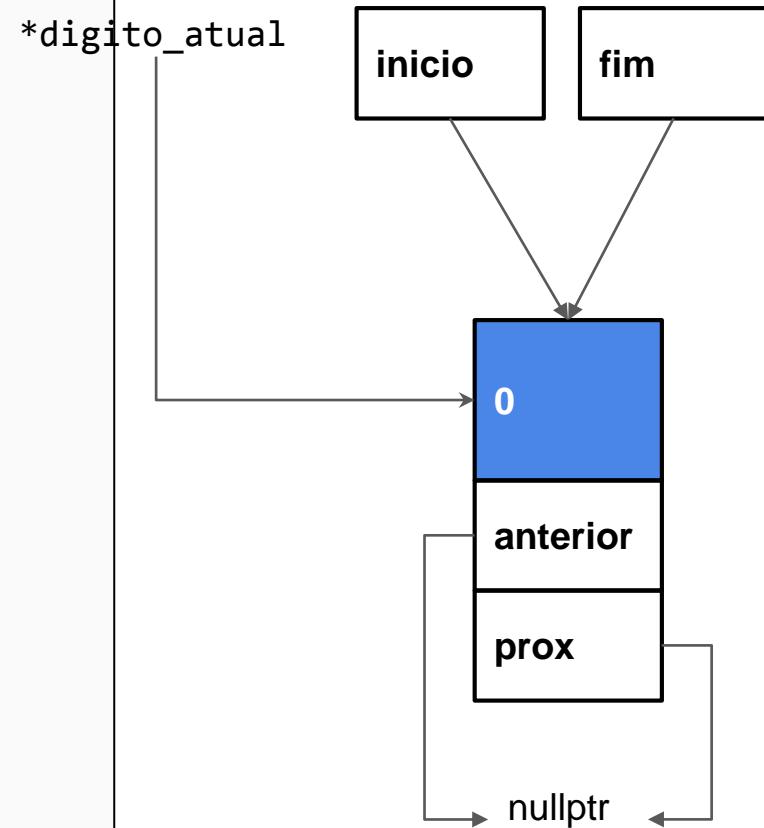
# Iniciando com 1 dígito zero

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



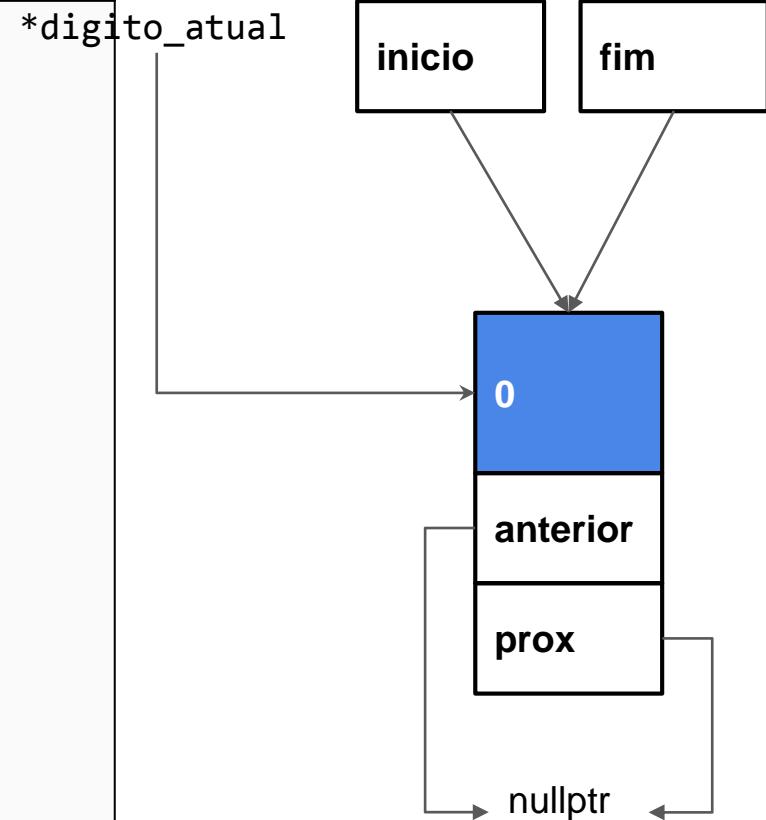
# Iniciando com 1 dígito zero

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



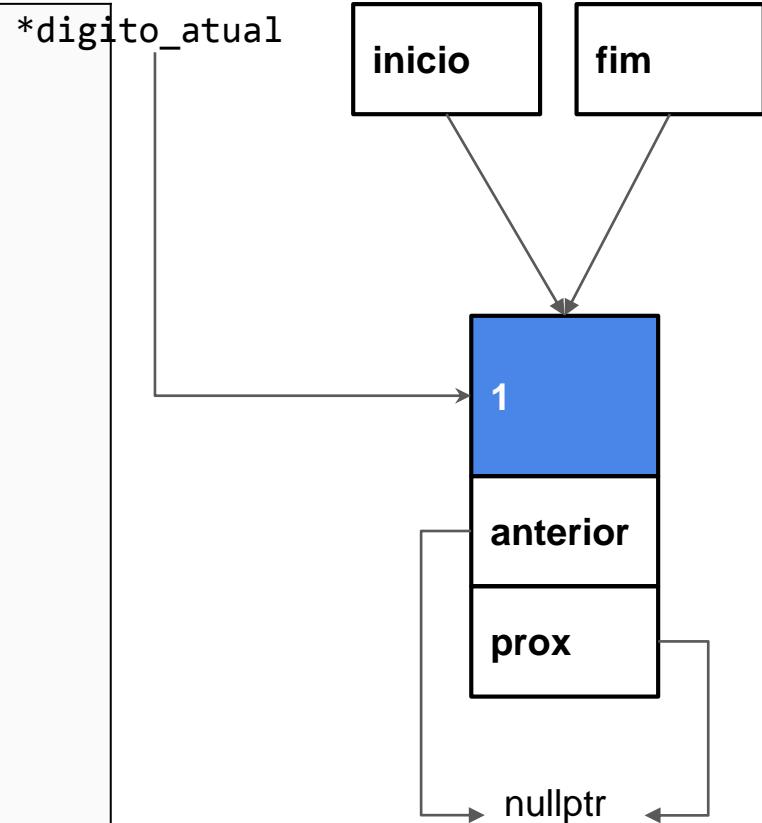
# valor < 9?

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



# ok → incrementa

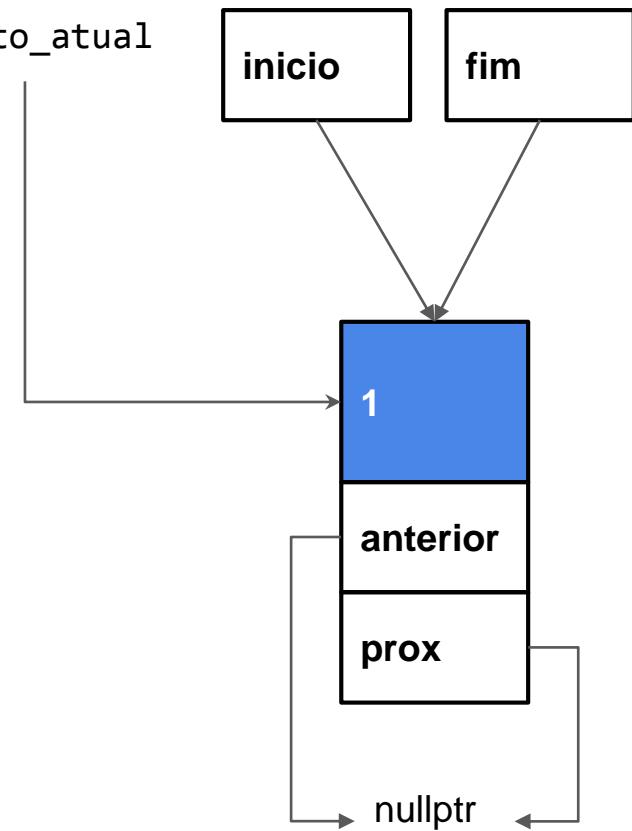
```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        ➔ if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



# vamos embora!

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```

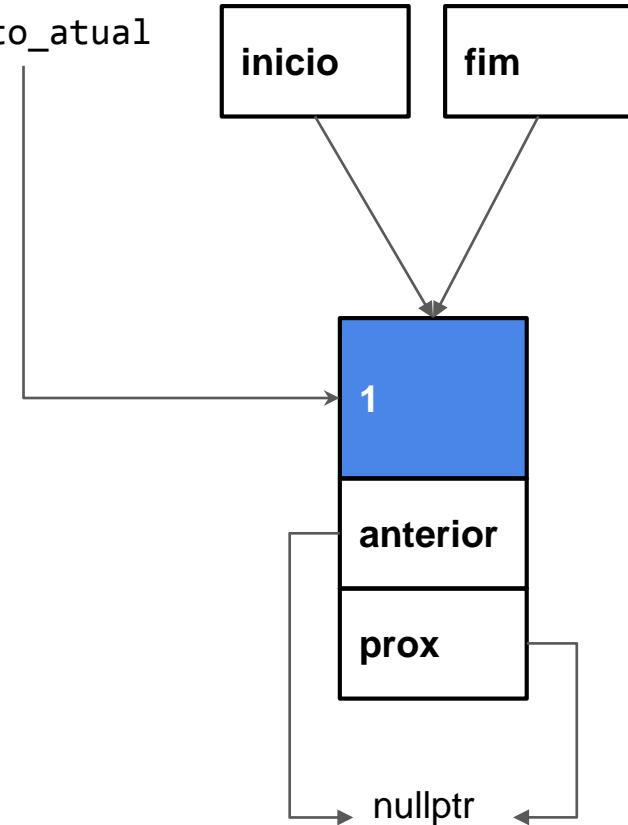
\*digito\_atual



# chamando + 1 vez a função incrementa

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```

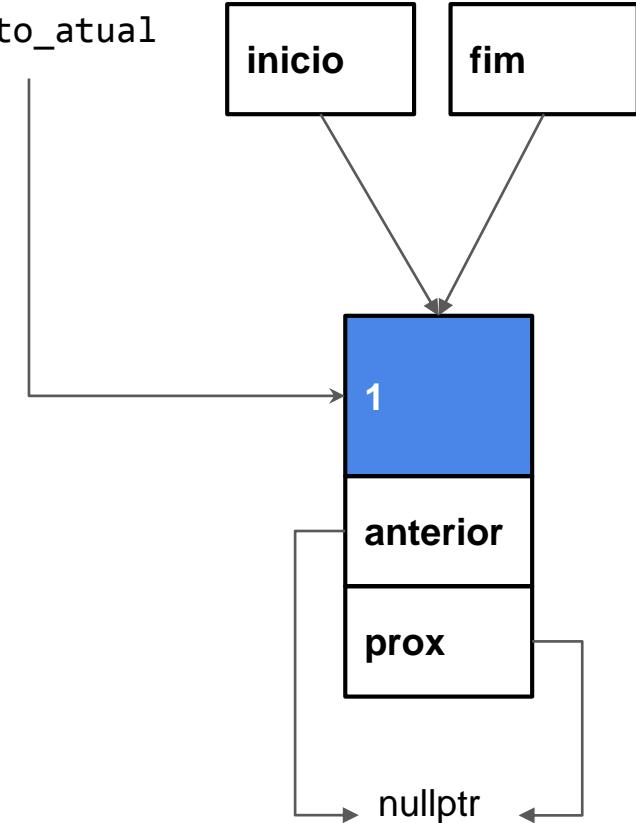
\*digito\_atual



# valor < 9?

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```

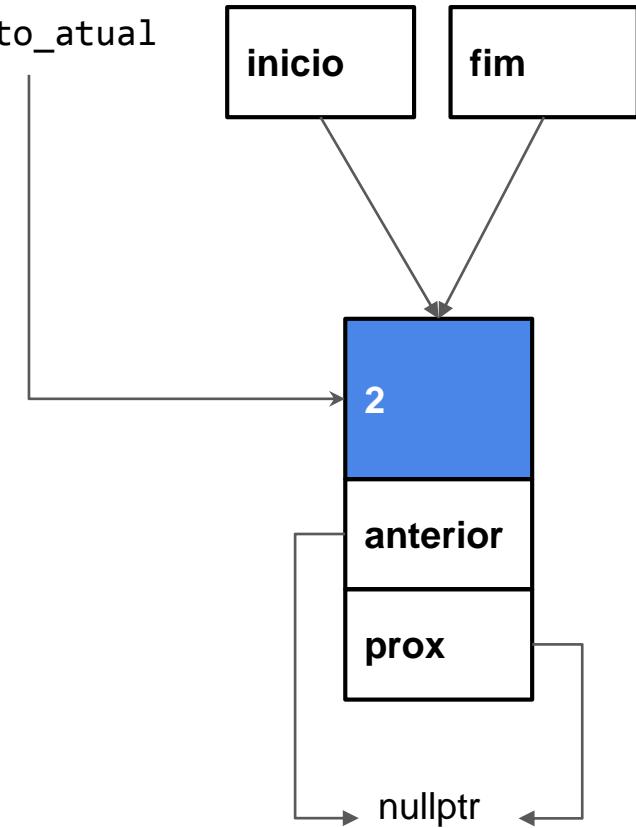
\*digito\_atual



# ok → incrementa

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```

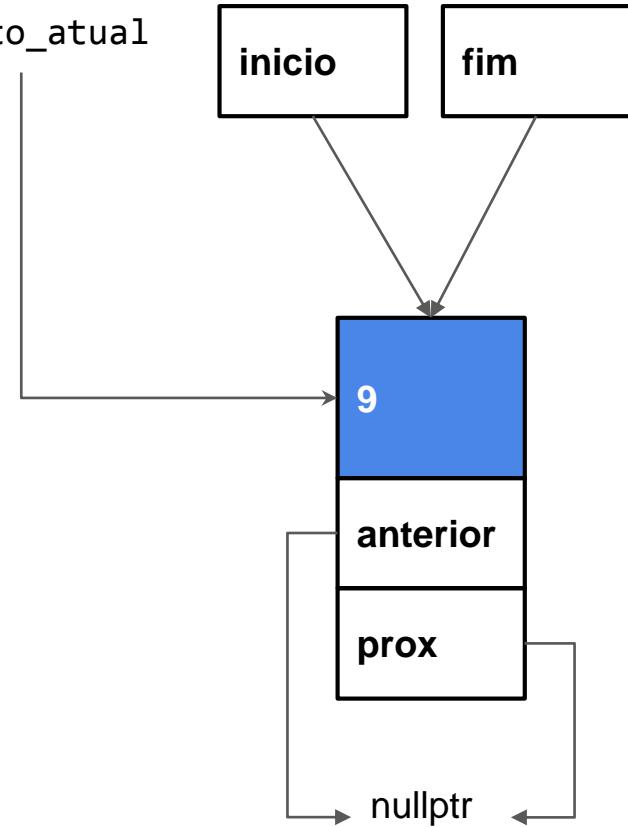
\*digito\_atual



# depois de 9 chamadas!

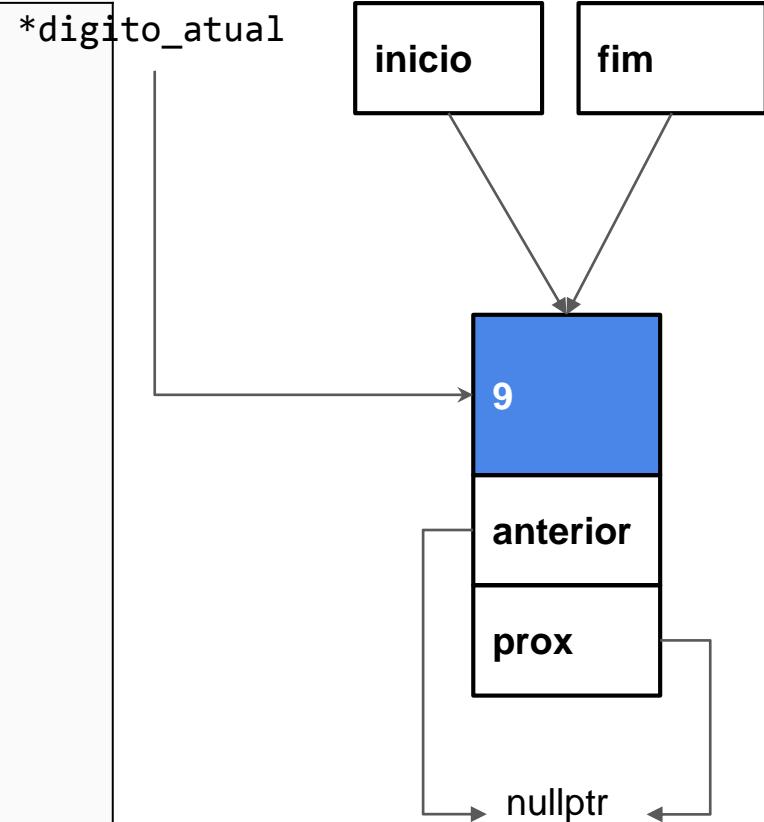
```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```

\*digito\_atual



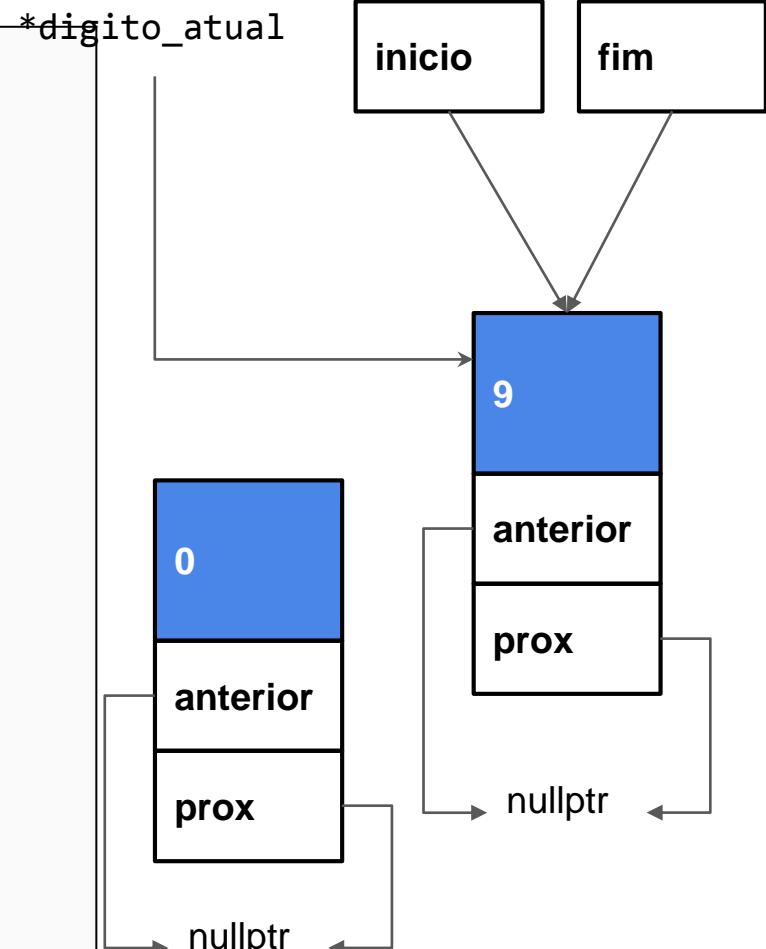
# < 9 → false

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



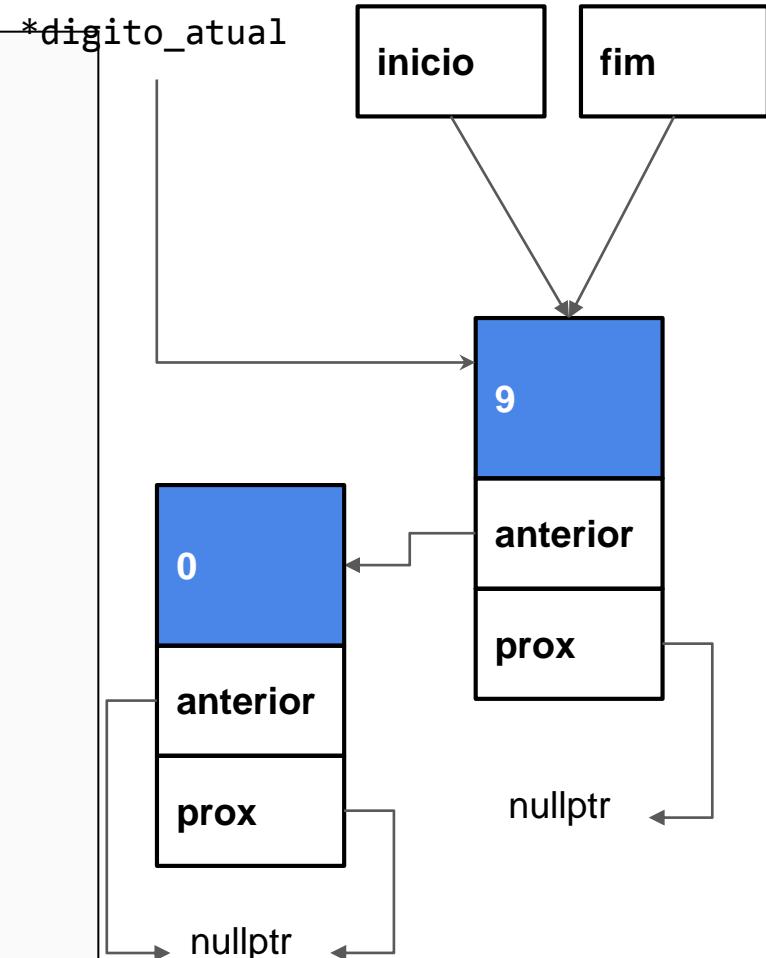
# novo dígito

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



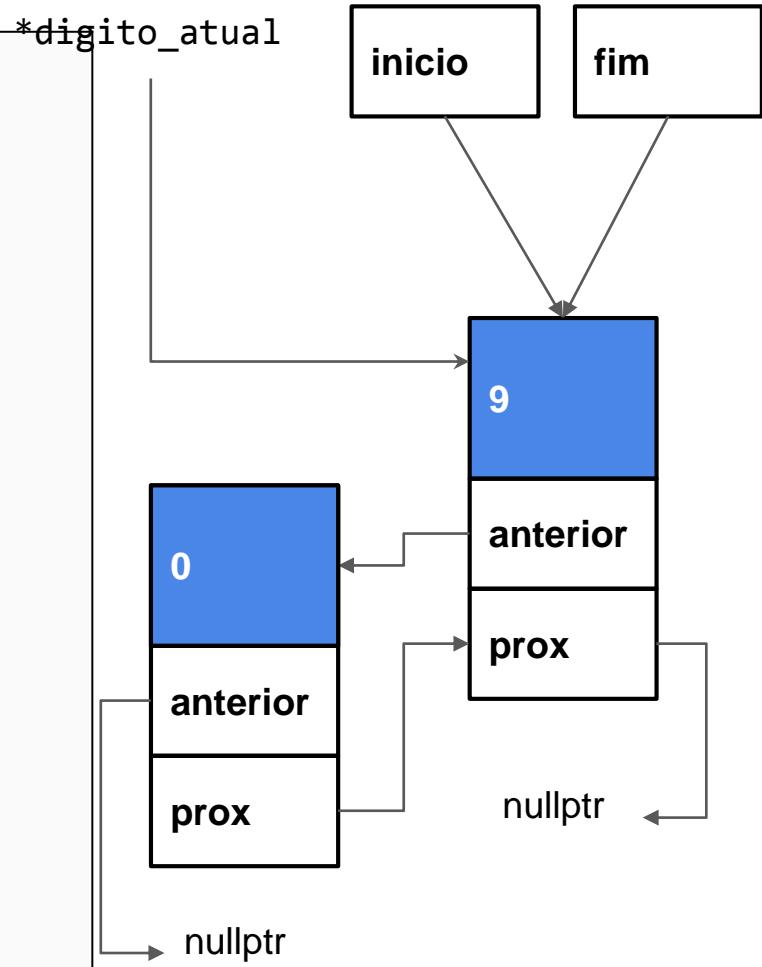
# próximo do atual

```
void BigNum::incrementa() {
    node_t *digito_atual = _fim;
    while (1) {
        if (digito_atual->valor < 9) {
            digito_atual->valor += 1;
            break;
        }
        if (digito_atual->anterior == nullptr) {
            digito_atual->anterior = new node_t();
            digito_atual->anterior->prox = digito_atual;
            _inicio = digito_atual->anterior;
            _inicio->prox = digito_atual;
        }
        digito_atual->valor = 0;
        digito_atual->anterior = nullptr;
        digito_atual = digito_atual->anterior;
    }
}
```



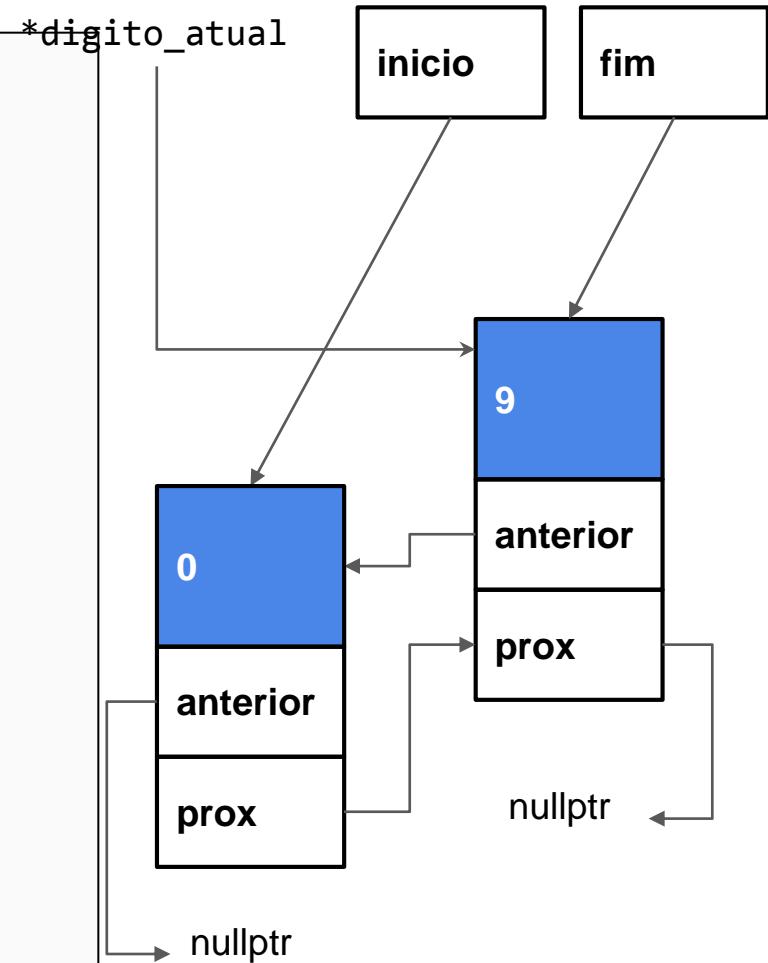
# tem como anterior o atual

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



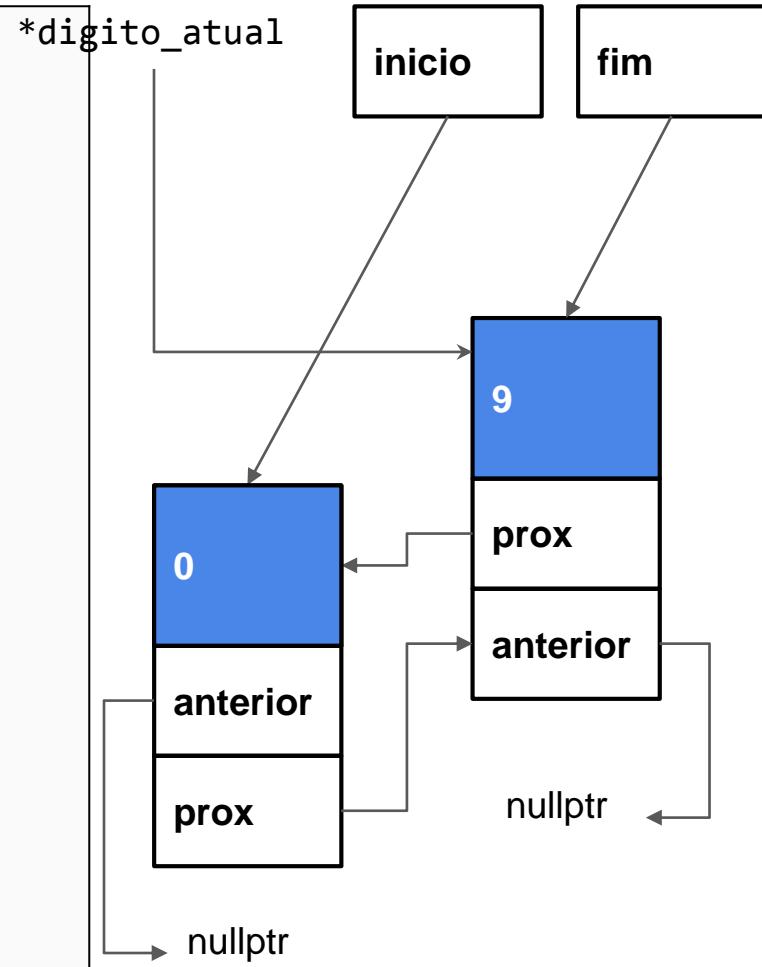
# atualizamos o início

```
void BigNum::incrementa() {
    node_t *digito_atual = _fim;
    while (1) {
        if (digito_atual->valor < 9) {
            digito_atual->valor += 1;
            break;
        }
        if (digito_atual->anterior == nullptr) {
            digito_atual->anterior = new node_t();
            digito_atual->anterior->proxima =
digito_atual;
            → _inicio = digito_atual->anterior;
            _inicio->proxima = digito_atual;
        }
        digito_atual->valor = 0;
        digito_atual->anterior = nullptr;
        digito_atual = digito_atual->anterior;
    }
}
```



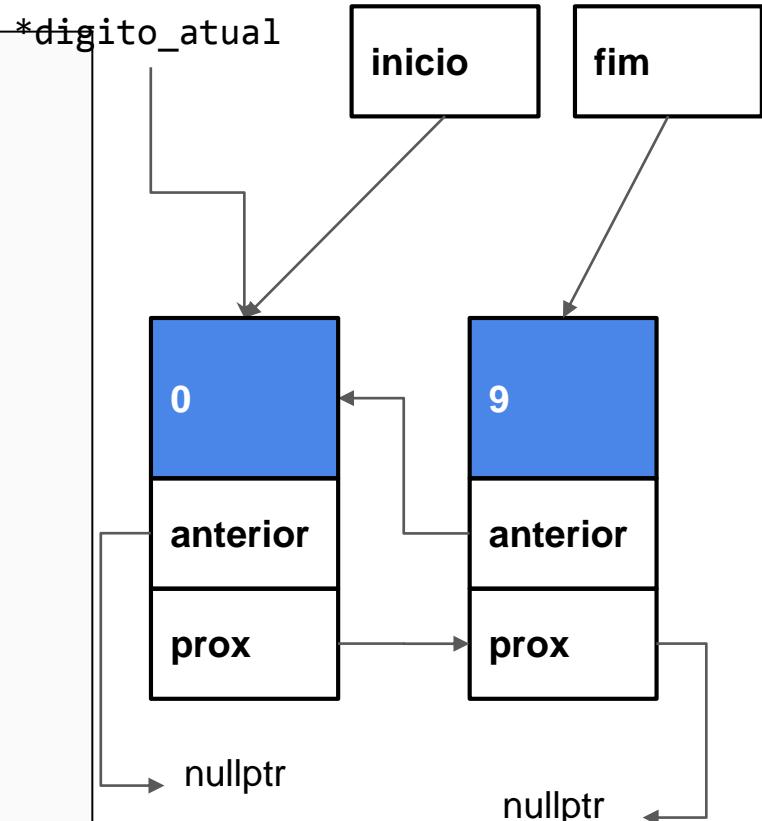
# zeramos o novo atual

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
            }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



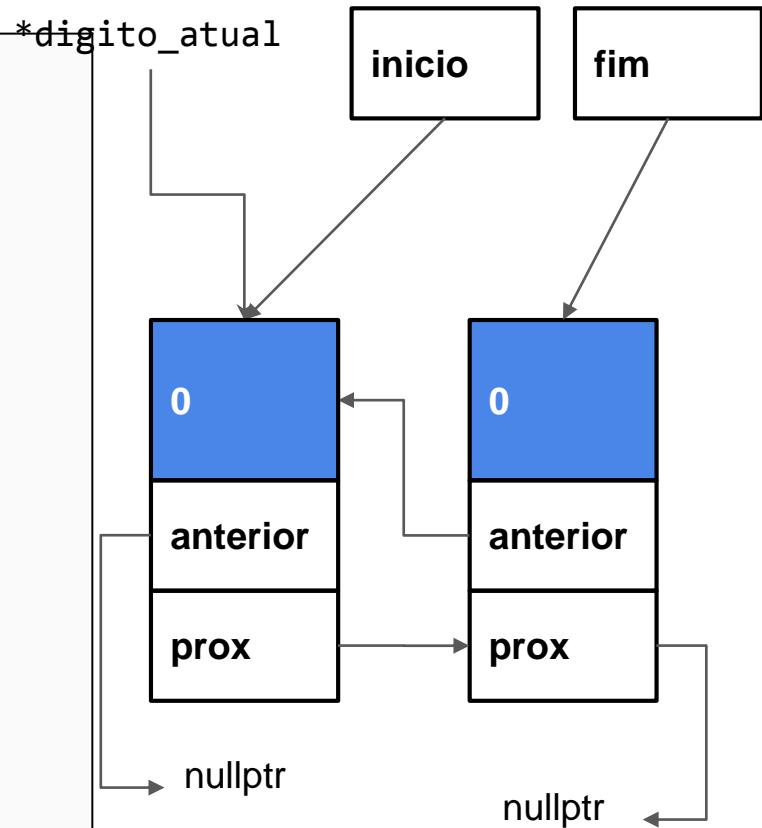
# caminha pra frente

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



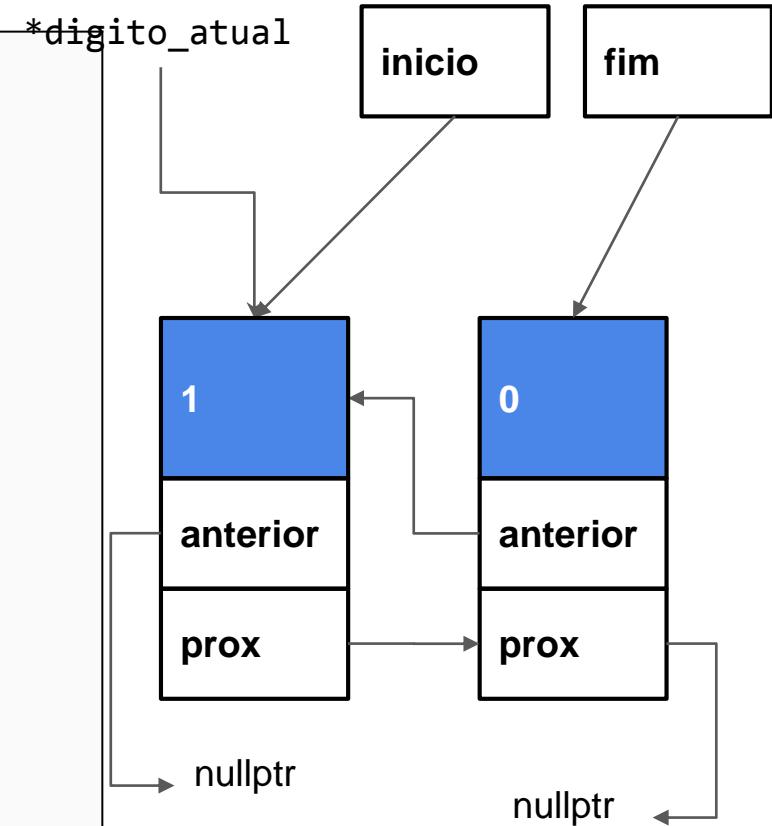
# e agora?

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



# chegamos no 10! note o while()

```
void BigNum::incrementa() {  
    node_t *digito_atual = _fim;  
    while (1) {  
        if (digito_atual->valor < 9) {  
            digito_atual->valor += 1;  
            break;  
        }  
        if (digito_atual->anterior == nullptr) {  
            digito_atual->anterior = new node_t();  
            digito_atual->anterior->proximo =  
digito_atual;  
            _inicio = digito_atual->anterior;  
            _inicio->proximo = digito_atual;  
        }  
        digito_atual->valor = 0;  
        digito_atual->anterior = nullptr;  
        digito_atual = digito_atual->anterior;  
    }  
}
```



# Até agora...

Quebra cabeça para próximos passos

- Structs
- Ponteiros e Referências
  - Ponteiros são mais utilizados internamente
  - Vide nossos TADs
- Precisamos fazer tudo do zero? Não! STL
- Podemos modelar comportamento comum? Sim! Interface.

# Programação e Desenvolvimento de Software 2

## Construção, depuração e revisão de código

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# Introdução

- Boas práticas de programação
  - Reduzem chance de erros (eles vão existir!)
  - Medidas proativas
  - Programação defensiva
  - Testes de unidade
- Meu programa não funciona! E agora?!
  - Vou re-escreer tudo do zero!
  - Medidas reativas
  - Depuração

## Escolha I

**Underscore**

```
struct TipoStruct {  
  
    int valor_da_conta;  
  
    int get_valor();  
    int set_valor();  
};
```

**CamelCase**

```
struct TipoStruct {  
  
    int valorDaConta;  
  
    int getValor();  
    int setValor();  
};
```

# Métodos e linhas

- Mantenha os métodos curtos
- Regra de uma ou meia tela
  - Se um método passa de uma, ou meia tela de monitor
  - Quebre em métodos menores
- Mantenha as linhas 80 ou 100 caracteres
  - Incluindo espaços

# Comentários

- Sempre que possível e necessário
- Devem
  - Ser informativos sobre o funcionamento
  - Alertar sobre possíveis consequências
  - Explicar o que a função faz, não como ela faz
- Não devem
  - Ser redundantes
  - Dizer algo que deveria estar claro no código

# Keep it Simple Stupid!

“One difference between a smart programmer and a professional programmer is that the professional understands that clarity is king. Professionals use their powers for good and write code that others can understand.”

— Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship

# Desenvolvimento voltado para testes

- Nos próximos laboratórios vamos iniciar o uso de testes de unidade
- Em particular, biblioteca doctest

<https://github.com/onqtam/doctest>

# Desenvolvimento voltado para testes

**Table 7-5. Hours to fix bug based on introduction point**

Stage Introduced	STAGE FOUND				
	Requirements	Coding/Unit Testing	Integration	Beta Testing	Post-product Release
Requirements	1.2	8.8	14.8	15.0	18.7
Coding/Unit testing	NA	3.2	9.7	12.2	14.8
Integration	NA	NA	6.7	12.0	17.3

NA = Not applicable because cannot find a bug before it is introduced

<https://blog.fullstory.com/what-we-learned-from-google-code-reviews-arent-just-for-catching-bugs/>

# **Depuração e Erros de Memória**

---

# Re-lembrando

## Tipos de erros

- Uma forma de ver:**
  - Erros de sintaxe
  - Erros de semântica
  - Erros de lógica
- Outra forma:**
  - Erros em tempo de compilação
  - Erros em tempo de execução

# Erros de Sintaxe

## Exemplos

- Quase sempre são erros de compilação

```
#include <iostream>

int main() {
    st::cout << "oi";
    return 0;
}
```

# Erros de Semântica

## Exemplos

- Podem ou não gerar um erro/warning

```
#include <iostream>

int *f() {
    int *rv[20];
    return rv;
}
```

# Erros de Lógica

## Não temos mais ajuda do compilador

```
#include <iostream>
#include <cmath>

int factorial(int num) {
    int fat = 0;
    for (int i = 1; i <= num; i++)
        fat = fat * i;
    return fat;
}

double series(double x, int n) {
    double valor = 0.0;
    double xpow = 1;
    for (int k = 0; k <= n; k++) {
        valor += xpow / factorial(k);
        xpow = xpow * x;
    }
    return valor;
}
```

# Erros de Lógica

## Não temos mais ajuda do compilador

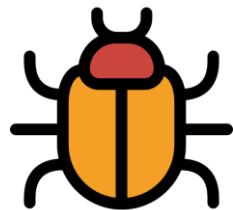
```
#include <iostream>
#include <cmath>

int factorial(int num) {
    int fat = 0; // Fatorial sempre zero
    for (int i = 1; i <= num; i++)
        fat = fat * i;
    return fat;
}

double series(double x, int n) {
    double valor = 0.0;
    double xpow = 1;
    for (int k = 0; k <= n; k++) {
        valor += xpow / factorial(k);
        xpow = xpow * x;
    }
    return valor;
}
```

# Depuração

## Motivação



- Encontrar erros em códigos é uma arte
- Com o tempo:
  - aprendemos a mapear comportamento anômalo para erros comuns
  - o depurador nos ajuda
- Vamos usar o GDB para o erro acima

# GNU Project Debugger

- GDB é o depurador “padrão” de C/C++
- Pode ser utilizado pela linha de comando
- Ou com uma GUI
  - <http://gdbgui.com>
- Ou em uma IDE
  - CodeBlocks e Visual Studio
- Até online
  - <https://www.onlinegdb.com/>

# Passos para Depurar o Código

- Reproduzir o problema
- Achar o local do erro
- Verificar efeitos colaterais
- Testar o código novamente
- Resolver

# Dicas Gerais

- Entenda as mensagens de erro
- Pense antes de escrever
- Procure por problemas comuns
- Dividir para conquistar
- Mostre o valor de variáveis importantes
- Utilize um depurador
- Concentre-se em mudanças recentes

# Dicas Ruins

- Encontrar defeitos adivinhando (na sorte)
- Fazer alterações aleatórias até funcionar
- Não fazer um backup do original e não manter um histórico das alterações feitas
- Corrigir o erro com a solução mais óvia sem entender a razão do problema
- Se o erro sumiu, tudo ok!

# Erros de Memória

- Memory Leaks
  - Código sem delete
- Acessos inválidos
  - Acesso para null
  - Acesso para memória desalocada
- Arquivos abertos
  - Arquivos sem close

# Erros de Memória

- Para erros de memória podemos fazer uso de ferramentas como
- Valgrind
  - Comum em ambientes Unix
- DrMemory
  - Disponível para Windows/Linux/Mac
- Dicas de como instalar:
  - <https://github.com/flaviovdf/programacao-2/tree/master/valgriddrmem>

# Ferramentas

---

# Processo de compilação

## Quando fazemos de forma manual

- Até o momento estamos compilando todo o código de uma vez
- Porém o mesmo é composto de módulos que são independentes
- Trabalhamos com tudo em uma pasta

```
$ g++ arq1.cpp arq2.cpp -o main
```

ou

```
$ g++ *.cpp -o main
```

# Formas de organizar o código

## Melhor organização final

### □ Agrupar o código em pastas lógicas

```
. raiz do projeto
|---- Makefile
|---- build/                               [diretório]
|      |---- [objetos compilados]
|---- include/                            [diretório]
|      |---- modulo1/                      [diretório]
|          |---- modulo1_arquivo1.h       [cabeçalho]
|          |---- modulo1_arquivo2.h       [cabeçalho]
|      |---- modulo2/                      [diretório]
|          |---- modulo2_arquivo1.h       [cabeçalho]
|          |---- modulo2_arquivo2.h       [cabeçalho]
|---- src/                                [diretório]
|      |---- modulo1/                      [diretório]
|          |---- modulo1_arquivo1.cpp     [código]
|          |---- modulo1_arquivo2.cpp     [código]
|      |---- modulo2/                      [diretório]
|          |---- modulo2_arquivo1.cpp     [código]
|          |---- modulo2_arquivo2.cpp     [código]
```

# Módulos + Namespaces

Úteis quando temos nomes repetidos

- Imagine uma biblioteca em C++ para diferentes tipos de jogos de carta
- Em todos os jogos temos:
  - Mão
  - Jogadores
  - Baralhos
- Porém as cartas são bem diferentes



# Organizando o Código

Note que temos definições repetidas

```
. raiz do projeto
|---- Makefile
|---- build/
|      |
|---- include/
|      |---- magic/
|          |---- carta.h
|          |---- uno/
|              |---- carta.h
|---- src/
|      |---- magic/
|          |---- carta.cpp
|      |---- uno/
|          |---- carta.cpp
```

[diretório]  
[objetos compilados]  
[diretório]  
[diretório]  
[cabeçalho]  
[diretório]  
[cabeçalho]  
[diretório]  
[diretório]  
[código]  
[diretório]  
[código]

# Namespaces

Uma ajuda com definições repetidas

- Situações como estas são comuns em um software grande
- Namespaces ajudam a resolver o problema
- Podemos ter nomes repetidos em namespaces diferentes

# Dois cartas.h, cartas.cpp

include/uno/carta.h

```
#ifndef PDS2_CARTAUNO_H
#define PDS2_CARTAUNO_H
namespace uno {
    enum Cor {AZUL, VERDE, AMARELO, VERMELHO};

    struct Carta {
        Cor _cor;
        int _numero;

        Carta(Cor cor, int numero);
        int get_numero() const;
        Cor get_cor() const;
    };
}
#endif
```

# Dois cartas.h, cartas.cpp

include/magic/carta.h

```
#ifndef PDS2_CARTAMAGIC_H
#define PDS2_CARTAMAGIC_H

#include <string>
namespace magic {
    struct Carta {

        std::string _nome;
        double _dano;

        Carta(std::string nome, double dano);
        double get_dano() const;
        std::string get_nome() const;
    };
}
#endif
```

# Implementação

- Note que indicamos qual namespace estamos implementando

```
#include "magic/carta.h"

using namespace magic;

Carta::Carta(std::string nome, double dano) {
    _nome = nome;
    _dano = dano;
}
double Carta::get_dano() const {
    return _dano;
}
std::string Carta::get_nome() const {
    return _nome;
}
```

- Diferenciamos os arquivos com o namespace. Estilo fazemos com o std

```
#include <string>

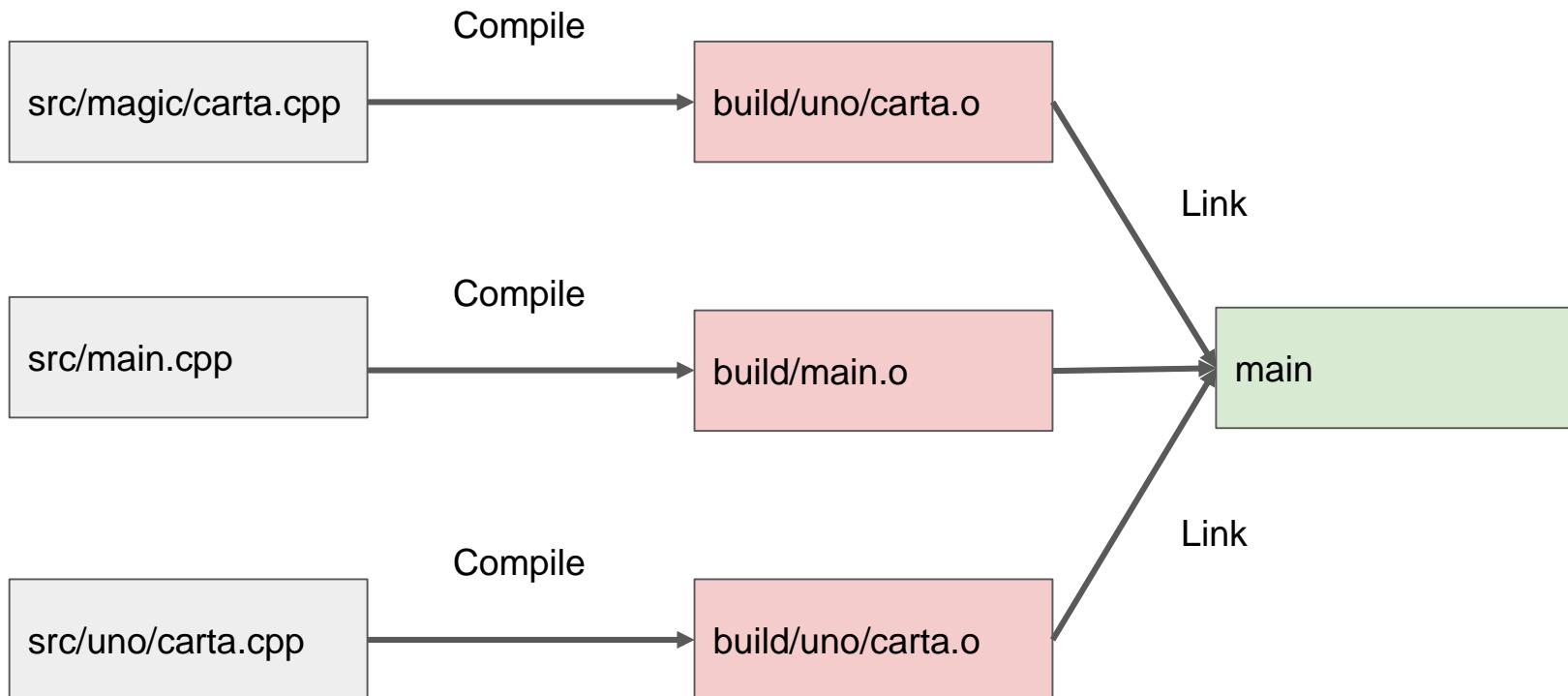
#include "magic/carta.h"
#include "uno/carta.h"

int main() {
    uno::Carta carta_uno(unocor::VERMELHO, 12);
    magic::Carta carta_magic("Monster", 72.1);
}
```

# Compilando Programas Maiores

Módulos podem ser compilados de forma independente

- Compilação ocorre em várias etapas
- Não observamos ao compilar tudo de uma vez



# Automatizando

## Makefiles

- Arquivos de compilação automática
- Indica as dependências entre os módulos
- Serve para automatizar desenvolvimento
- Conjunto de regras e dependências

```
target1 target2 ... : dependencial dependencia2 ...
    <TAB> comando1
    <TAB> comando2
    ...
```

```
helloworld: helloworld.cpp
    g++ -o helloworld helloworld.cpp
```

# Makefile

```
CC := g++
SRCDIR := src
BUILDDIR := build
TARGET := main
CFLAGS := -g -Wall -O3 -std=c++11 -I include/

all: main

magic:
    @mkdir build/magic/
    $(CC) $(CFLAGS) -c src/magic/carta.cpp -o build/magic/carta.o

uno:
    @mkdir build/uno/
    $(CC) $(CFLAGS) -c src/uno/carta.cpp -o build/uno/carta.o

main: magic uno
    $(CC) $(CFLAGS) build/magic/carta.o build/uno/carta.o src/main.cpp -o
main

clean:
    $(RM) -r $(BUILDDIR)/* $(TARGET)
```

# Makefile

- Mais um código para manter
- É possível fazer Makefiles mais genéricos
  - Funcionam em projetos que seguem a mesma organização
- Exemplo no Github

# Programação e Desenvolvimento de Software 2

## TADs específicos (STL)

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# Introdução

- Nenhum programa é escrito em uma linguagem de programação a partir do zero
- Geralmente
  - Linguagens vêm com bibliotecas
  - Impossível decorar todas
  - Usamos a documentação para entender
- Bibliotecas podem ser vistas como:
  - Conjunto de TADs e funções de uso geral

# Exemplos de TADs

- Coleções/Containers
  - Listas, Árvores
- Números
  - Bignum
  - Complexo
- Geometria
  - Ponto

# TADs do dia a dia

- Existem na biblioteca padrão de C++
- Para PDS2
  - Não precisamos ir muito além da padrão
- No pior dos casos, tente na Boost
  - <https://www.boost.org/>
  - Bignums e números complexos

# Bibliotecas v. Frameworks

## □ Bibliotecas

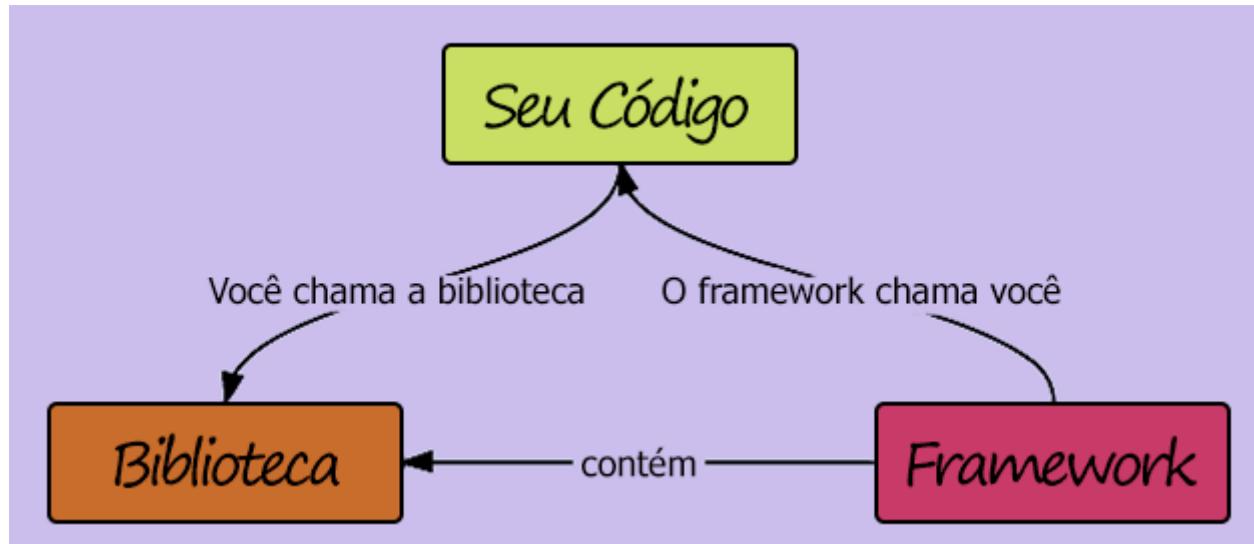
- Funcionalidades mais comuns
- Containers, aritmética, matemática etc

## □ Frameworks

- Servem para um propósito maior
- Serviços web
- Engines de jogos já prontas

# Uma forma de ver

- A regra abaixo não vale sempre
- Mas se existir muitos callbacks
  - Chance de ser um framework



# Componentes da biblioteca padrão

- Biblioteca padrão C++
- Strings
  - Suporte para expressões regulares
- Ponteiros inteligentes para gerenciamento de recurso (e.g. unique\_ptr e shared\_ptr)
- Um framework de containers (e.g. vector e map) e algoritmos (e.g. find() e sort())
- Convencionalmente chamado STL  
(Standard Template Library)

# Headers da Biblioteca Padrão

- Qualquer funcionalidade da biblioteca padrão é fornecida através de um header padrão:

```
#include <string>
```

```
#include <vector>
```

- A biblioteca padrão é definida em um namespace chamado std. Para usar as funcionalidades, o prefixo std:: é usado:

```
std::string gato = "O gato miou.;"
```

```
std::vector<std::string> palavras =  
{"gato", "pi"};
```

# Headers da Biblioteca Padrão

- Por simplicidade, podemos evitar o uso de std::
- Geralmente não é uma boa prática carregar todos os nomes de um namespace no namespace global

```
#include <string>
using namespace std;
string s = "O gato miou.";
```

- Explícito é melhor!

```
#include <string>
std::string s = "O gato miou.";
```

# Strings em C++

## □ Funções no nível da biblioteca

<http://www.cplusplus.com/reference/string>

### *fx* Functions

#### Convert from strings

<a href="#">stoi</a> <small>C++11</small>	Convert string to integer ( <a href="#">function template</a> )
<a href="#">stol</a> <small>C++11</small>	Convert string to long int ( <a href="#">function template</a> )
<a href="#">stoul</a> <small>C++11</small>	Convert string to unsigned integer ( <a href="#">function template</a> )
<a href="#">stoll</a> <small>C++11</small>	Convert string to long long ( <a href="#">function template</a> )
<a href="#">stoull</a> <small>C++11</small>	Convert string to unsigned long long ( <a href="#">function template</a> )
<a href="#">stof</a> <small>C++11</small>	Convert string to float ( <a href="#">function template</a> )
<a href="#">stod</a> <small>C++11</small>	Convert string to double ( <a href="#">function template</a> )
<a href="#">stold</a> <small>C++11</small>	Convert string to long double ( <a href="#">function template</a> )

#### Convert to strings

<a href="#">to_string</a> <small>C++11</small>	Convert numerical value to string ( <a href="#">function</a> )
<a href="#">to_wstring</a> <small>C++11</small>	Convert numerical value to wide string ( <a href="#">function</a> )

# Exemplo

## □ **to\_string:** Numérico → String

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

int main(void) {
    int valor = 0;
    std::cin >> valor;
    std::string valor_como_texto = "O valor foi: " + std::to_string(valor);
    std::cout << valor_como_texto;
}
```

# Standard Template Library

Templates indicam um tipo genérico

- Programação Genérica
  - A mesma definição de função atua da mesma forma sobre objetos de diferentes tipos
- Consiste de um tipo de Polimorfismo
  - Vamos aprender sobre outros ao usar herança e interfaces
- Templates (C++), Generics (Java)

# Standard Template Library

Templates indicam um tipo genérico

```
#ifndef PDS2_LISTAGENERICA_H
#define PDS2_LISTAGENERICA_H

template <typename T>
struct node_t {
    T elemento;
    node_t *proxima;
};

template <typename T>
struct ListaSimplesmenteEncadeada {

    node_t<T> *_início;
    node_t<T> *_fim;
    int _num_elementos_inseridos;

    ListaSimplesmenteEncadeada();
    ~ListaSimplesmenteEncadeada();
    void inserir_elemento(T elemento);
    void imprimir();
};

#endif
```

# Standard Template Library

Templates indicam um tipo genérico

```
#ifndef PDS2_LISTAGENERICA_H
#define PDS2_LISTAGENERICA_H

template <typename T> // Template para qualquer struct/class
struct node_t {
    T elemento;
    node_t *proxima;
};

template <typename T> // Temos que definir para cada struct/class
struct ListaSimplesmenteEncadeada {

    node_t<T> *_início;
    node_t<T> *_fim; // Aqui dizemos que node<T> usa T de ListaSimplesmente...
    int _num_elementos_inseridos;

    ListaSimplesmenteEncadeada();
    ~ListaSimplesmenteEncadeada();
    void inserir_elemento(T elemento);
    void imprimir();
};

#endif
```

# Standard Template Library

Templates indicam um tipo genérico

## □ Fazendo uso

```
#include <string>

#include "listasimples.h"

int main(void) {
    ListaSimplesmenteEncadeada<int> lista = ListaSimplesmenteEncadeada<int>();
    for (int i = 0; i < 1000; i++)
        lista.inserir_elemento(i);
    lista.imprimir();

    ListaSimplesmenteEncadeada<std::string> lista2 = \
        ListaSimplesmenteEncadeada<std::string>();
    lista2.inserir_elemento("flavio");
    lista2.inserir_elemento("douglas");
    lista2.imprimir();
    return 0;
}
```

# Standard Template Library

Templates indicam um tipo genérico

- Exemplo de implementação no Github
  - Para quem quiser criar templates
  - Não vamos nos preocupar tanto em como implementar, sim como fazer uso

# Standard Template Library

## Containers

- Coleções de objetos
- Uso de containers apropriados para uma tarefa e suportá-los com operações fundamentais é crucial
- Containers usam templates por baixo
  - Assim fazemos uso de qualquer tipo
- Nem sempre o mesmo container é o melhor para diferentes problemas

# Containers

## Sequenciais

- Vector
- Deque
- List

## Associativos

- Set
- Map
- Multiset
- Multimap

## Adaptadores

- Stack
- Queue
- Priority queue

# Containers

## □ Fazendo elo com os TADs

### Sequence Containers:

Array:



Vector:



Deque:



List:

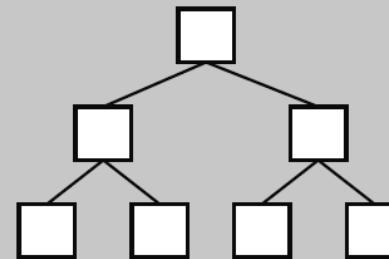


Forward-List:

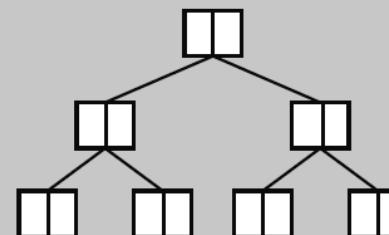


### Associative Containers:

Set/Multiset:

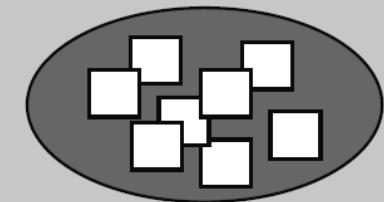


Map/Multimap:

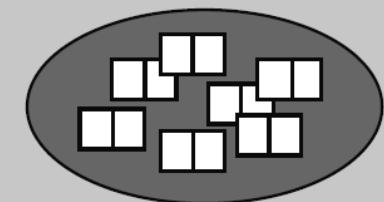


### Unordered Containers:

Unordered Set/Multiset:



Unordered Map/Multimap:



# vector

- Um dos containers mais úteis é o vector
- Um vector é uma sequência de elementos
- Por baixo é uma lista com array

# vector: uso

- Vamos iniciar com uma classe Pessoa
- Já implementada

```
#include <string>

struct Pessoa {

    const std::string _nome;
    int _idade;

    // Construtor com Lista de inicialização
    Pessoa(std::string nome, int idade):
        _nome(nome), _idade(idade) {}

    std::string get_nome() const {
        return this->_nome;
    }

    int get_idade() const {
        return this->_idade;
    }
};
```

No nosso caso `_nome` nunca muda, const

Primeira vez que vemos esse construtor.  
Funciona igual ao anterior. Necessário const

Métodos const nunca mudam o objeto. Garantido.

# vector: uso

- Similar aos nossos TADs
- `push_back` → Inserção; `at` → Acesso

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

#include "pessoa.h"

int main() {
    std::vector<Pessoa> pessoas;
    pessoas.push_back(Pessoa("Ana", 18));
    pessoas.push_back(Pessoa("Pedro", 19));

    // Primeira forma de acesso
    std::cout << pessoas[0].get_nome() << std::endl;
    std::cout << pessoas[1].get_nome() << std::endl;

    // Segunda forma, com at
    std::cout << pessoas.at(0).get_nome() << std::endl;
    std::cout << pessoas.at(1).get_nome() << std::endl;
    return 0;
}
```

# vector de inteiros

```
#include <iostream>
#include <vector>

int main() {
    std::vector<int> v = {7, 5, 16, 8};
    v.push_back(25);
    v.push_back(13);

    for(int n : v) { ← Iterador for each. Percorre todos os elementos
        std::cout << n << std::endl;
    }

    return 0;
}
```

□ for each: equivale ao laço abaixo

```
for (int i = 0; i < v.size(); i++) int n = v[i];
```

- Um **vector** pode ser copiado:

```
std::vector<Pessoa> lista2 = lista_tel;
```

- Atribuir um **vector** envolve copiar seus elementos.  
Após a inicialização de lista2, lista\_tel e lista2 têm  
cópias separadas de cada elemento
- Tal inicialização pode ser cara
- Quando a cópia é indesejável, referências e  
ponteiros devem ser utilizados

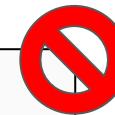
# Qual o problema das chamadas abaixo?

```
void ano_novo(std::vector<Pessoa> pessoas) {
    for (Pessoa pessoa : pessoas)
        pessoa.set_idade(pessoa.get_idade() + 1);
}
```

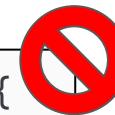
```
std::vector<std::string> pegar_nomes(std::vector<Pessoa> pessoas) {
    std::vector<std::string> nomes;
    for (Pessoa pessoa : pessoas)
        nomes.push_back(pessoa.get_nome());
    return nomes;
}
```

# Qual o problema das chamadas abaixo?

```
void ano_novo(std::vector<Pessoa> pessoas) {  
    for (Pessoa pessoa : pessoas)  
        pessoa.set_idade(pessoa.get_idade() + 1);  
}
```



```
std::vector<std::string> pegar_nomes(std::vector<Pessoa> pessoas) {  
    std::vector<std::string> nomes;  
    for (Pessoa pessoa : pessoas)  
        nomes.push_back(pessoa.get_nome());  
    return nomes;  
}
```



- (1) Passagem por cópia
- (2) Mesmo se fosse por referência, for each faz cópia

# Forma correta

```
// Sem const, vamos mudar a memória (cada pessoa aumenta de idade)
void ano_novo(std::vector<Pessoa> &pessoas) {
    for (int i = 0; i < pessoas.size(); i++)
        pessoas.at(i).set_idade(pessoas.at(i).get_idade() + 1);
}

// Com const, leitura apenas.
std::vector<std::string> pegar_nomes(std::vector<Pessoa> const &pessoas) {
    std::vector<std::string> nomes;
    for (Pessoa pessoa : pessoas)
        nomes.push_back(pessoa.get_nome());
    return nomes;
}
```

- Note o uso de **const** quando apenas lemos
- Note que não usamos `for each` quando alteramos os objetos. Laço normal.

# Diferentes laços

Assumindo um vetor de inteiros, explique cada caso

## □ Laço clássico

```
std::vector<int> dados = {0, 7, 8, 1, 3};  
for (int i = 0; i < dados.size(); i++)  
    std::cout << dados[i];
```

## □ Laço compacto

```
for (int x : dados)  
    std::cout << x;
```

## □ Laço para a referência

```
for (int &x : dados)  
    x *= 2;
```

- Lista duplamente encadeada
- Não temos mais acesso via índice. Motivo?
  - Iterador para acessar os elementos

```
#include <iostream>
#include <list>

int main() {
    std::list<int> l = {7, 5, 16, 8};

    // Add an integer to the front of the list
    l.push_front(25);
    // Add an integer to the back of the list
    l.push_back(13);

    for (std::list<int>::iterator it=l.begin(); it != l.end(); ++it) {
        std::cout << *it << std::endl;
    }
    return 0;
}
```

# list

## Sobre iteradores

- Funcionam de forma similar a ponteiros
- Lembre-se da aritmética de ponteiros

```
#include <iostream>
#include <list>

int main() {
    std::list<int> l = {7, 5, 16, 8};

    // Add an integer to the front of the list
    l.push_front(25);
    // Add an integer to the back of the list
    l.push_back(13);

    for (std::list<int>::iterator it=l.begin(); it != l.end(); ++it) {
        std::cout << *it << std::endl;
    }
    return 0;
}
```

# Iteradores

- Geralmente não acessamos elementos usando índices quando usamos uma lista encadeada
- Quando queremos identificar um elemento em uma list usamos um iterador
- Todo container da biblioteca padrão oferece as funções begin() e end(), que retorna um iterador pro primeiro e depois do último elemento

# Iteradores

São basicamente ponteiros (pelo menos em C++)

- `l.begin()` → ponteiro para primeiro elemento
- `l.begin() + 1` → ponteiro para segundo

```
#include <iostream>
#include <list>

int main() {
    std::list<int> l = {7, 5, 16, 8};

    // Add an integer to the front of the list
    l.push_front(25);
    // Add an integer to the back of the list
    l.push_back(13);

    for (std::list<int>::iterator it=l.begin(); it != l.end(); ++it) {
        std::cout << *it << std::endl;
    }
    return 0;
}
```

# Iteradores

## □ Usando um iterador:

```
for (std::list<int>::iterator it=l.begin(); it != l.end(); ++it) {  
    std::cout << *it << std::endl;  
}
```

## □ Dado um iterador `it`, `*it` é o elemento que ele se refere, `++p` avança p para o próximo elemento

# list vs vector

- Quando queremos uma sequência de elementos, podemos escolher entre `vector` e `list`
- A não ser que tenha um motivo, use `vector`, ele tem desempenho melhor para percorrer (e.g., `find()`), e para ordenar e pesquisar (e.g., `sort()`)

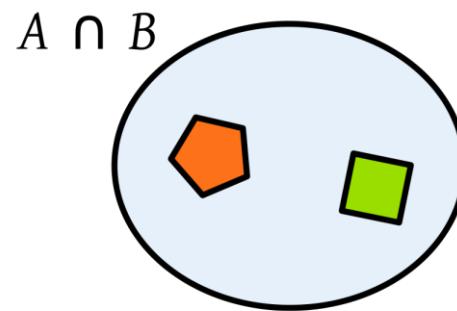
# set

## Conjuntos matemáticos

- Elementos únicos
- Sem ordem
- Embora podemos representar ordenado

$$A = \{\text{pentagono laranja}, \text{diamante azul}, \text{quadrado verde}, \text{retângulo amarelo}\}$$

$$B = \{\text{estrela vermelha}, \text{quadrado verde}, \text{triângulo verde}, \text{pentagono laranja}\}$$



# set: uso

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s;
    for(int i = 1; i <= 10; i++) {
        s.insert(i);
    }

    std::cout << "(" << s.size() << ")" << std::endl;
    for (int e : s) {
        std::cout << e << std::endl;
    }

    s.insert(7);

    std::cout << "(" << s.size() << ")" << std::endl;
    for (int e : s) {
        std::cout << e << std::endl;
    }
    for(int i = 2; i <= 10; i += 2) {
        s.erase(i);
    }

    std::cout << "(" << s.size() << ")" << std::endl;
    for (int e : s) {
        std::cout << e << std::endl;
    }
    return 0;
}
```

# set: uso

```
#include <iostream>
#include <set>

int main() {
    std::set<int> s;
    for(int i = 1; i <= 10; i++) {
        s.insert(i);
    }

    std::cout << "(" << s.size() << ")" << std::endl;
    for (int e : s) {
        std::cout << e << std::endl;
    }
}

s.insert(7);

std::cout << "(" << s.size() << ")" << std::endl;
for (int e : s) {
    std::cout << e << std::endl;
}
for(int i = 2; i <= 10; i += 2) {
    s.erase(i);
}

std::cout << "(" << s.size() << ")" << std::endl;
for (int e : s) {
    std::cout << e << std::endl;
}
return 0;
}
```

s = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

s = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

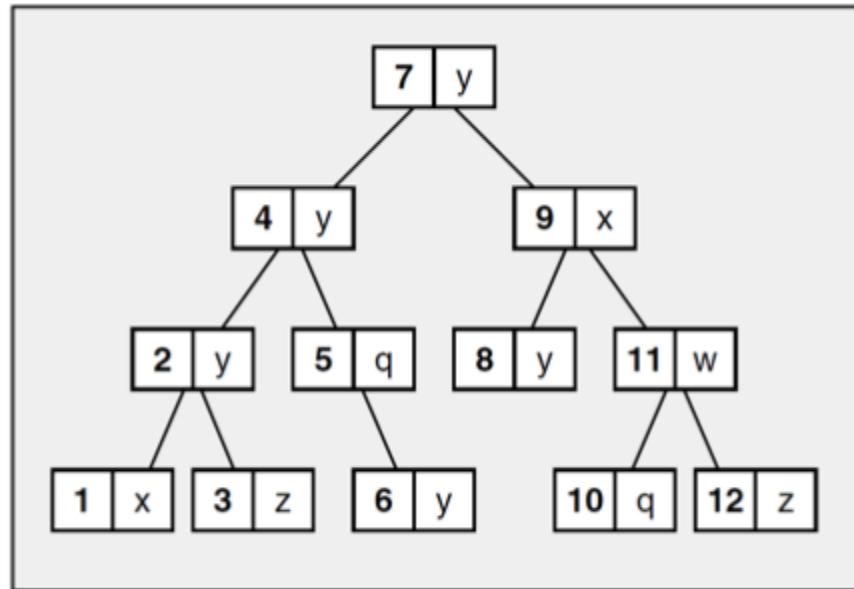
s = {1, 3, 5, 7, 9}

- Dicionário de chave → valor
- Uma árvore por baixo
  - Especial: sempre balanceada
  - Guarda pares de elementos

# map

## entendendo a memória por baixo

- Cada elemento é um nó
- Guarda uma chave (número neste caso)
- Valor (letra neste caso)



# map

## exemplo

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

int main() {
    std::map<int, std::string> m;
    m.insert(std::pair<int, std::string>(2017123456, "Joao"));

    m[2016123456] = "Maria";
    m[2018123456] = "Carlos";
    m[2015123456] = "Jose";
    m[2014123456] = "Joana";

    std::map<int, std::string>::iterator it;
    for (it = m.begin(); it != m.end(); it++) {
        std::cout << it->first << ":" << it->second << std::endl;
    }
    return 0;
}
```

# Versões sem ordem

## unordered\_map e unordered\_set

- Por padrão, maps/sets são implementados como árvores binárias de busca
- Existem versões `unordered_*`
  - Mais eficazes na prática
  - Porém, não conseguimos ordenar as chaves
- Iterador em um map/set
  - Sempre em ordem

# Criando mapas de tipos diferentes

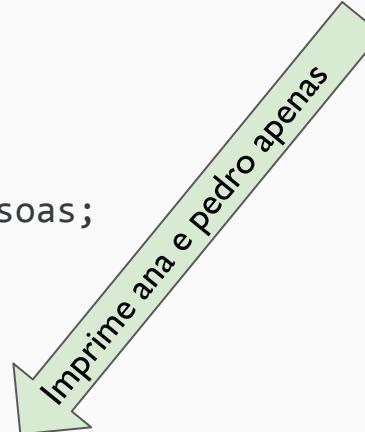
Precisamos saber comparar. Usamos um struct comparator

```
// O comparator sempre verificar se é <. Com < podemos criar >, == e != . Note que:  
//           p1 < p2  <-> p2 > p1  
//           p1 >= p2 <-> !(p1 < p2)  
//           p1 == p2 <-> !(p1 < p2) && !(p2 < p1)  
//           p1 != p2 <-> !(p1 == p2)  
struct compara_pessoa_f {  
    bool operator()(const Pessoa& p1, const Pessoa& p2) {  
        return p1.get_idade() < p2.get_idade();  
    }  
};  
  
int main() {  
    std::set<Pessoa, compara_pessoa_f> pessoas;  
    pessoas.insert(Pessoa("Ana", 18));  
    pessoas.insert(Pessoa("Pedro", 19));  
    pessoas.insert(Pessoa("Ana", 18));  
  
    for (Pessoa p : pessoas)  
        std::cout << p.get_nome() << std::endl;  
  
    return 0;  
}
```

# Criando mapas de tipos diferentes

Precisamos saber comparar. Usamos um struct comparator

```
// O comparator sempre verificar se é <. Com < podemos criar >, == e != . Note que:  
//           p1 < p2  <-> p2 > p1  
//           p1 >= p2 <-> !(p1 < p2)  
//           p1 == p2 <-> !(p1 < p2) && !(p2 < p1)  
//           p1 != p2 <-> !(p1 == p2)  
struct compara_pessoa_f {  
    bool operator()(const Pessoa& p1, const Pessoa& p2) {  
        return p1.get_idade() < p2.get_idade();  
    }  
};  
  
int main() {  
    std::set<Pessoa, compara_pessoa_f> pessoas;  
    pessoas.insert(Pessoa("Ana", 18));  
    pessoas.insert(Pessoa("Pedro", 19));  
    pessoas.insert(Pessoa("Ana", 18));  
  
    for (Pessoa p : pessoas)  
        std::cout << p.get_nome() << std::endl;  
  
    return 0;  
}
```



Imprime ana e Pedro apenas

# Outras bibliotecas de C++

<algorithm>	copy(), find(), sort()
<cmath>	sqrt(), pow()
<fstream>	fstream, ifstream, ofstream
<iostream>	istream, ostream, cin, cout
<memory>	unique_ptr, shared_ptr

# Programação e Desenvolvimento de Software 2

## Resolvendo Problemas com TADs

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# Problema 00

- Dada uma TAD Aluno
- Data um arquivo de Nome, Aluno, Matéria
- Como:
  - Alocar um novo aluno para cada registro (linha)

# Classe Aluno

```
#ifndef PDS2_ALUNO_H
#define PDS2_ALUNO_H

#include <string>

struct Aluno {

    std::string _nome;
    int _matricula;

    Aluno(std::string nome, int matricula);
    std::string get_nome();
    int get_matricula();
};

#endif
```

# Solução

Usando a entrada padrão (`std::cin` ou teclado)

```
#include <iostream>
#include <sstream>
#include <string>
#include "aluno.h"

int main() {
    std::string linha;
    std::istringstream stream_string;

    // 1. Lê o stream cin (padrão/teclado) Linha a Linha
    while (std::getline(std::cin, linha)) {
        if (linha == "sair") // 2. Se o usuário digitar sair, quebra o Laço
            break;
        // 3. Separa a Linha em palavra. Para isto, se usa o istringstream
        // Similar ao sscanf de C, usamos um texto como um arquivo.
        stream_string = std::istringstream(linha);
        std::string nome;
        int matricula;
        int codigo_disciplina;
        stream_string >> nome;
        stream_string >> matricula;
        stream_string >> codigo_disciplina;

        Aluno aluno(nome, matricula);
    }
}
```

# Solução

## Usando a entrada padrão (`std::cin` ou teclado)

```
#include <iostream>
#include <sstream>
#include <string>
#include "aluno.h"

int main() {
    std::string linha;
    std::istringstream stream_string;

    // 1. Lê o stream cin (padrão/teclado) linha a linha
    while (std::getline(std::cin, linha)) {
        if (linha == "sair") // 2. Se o usuário digitar sair, quebra o Laço
            break;
        // 3. Separa a Linha em palavra. Para isto, se usa o istringstream
        // Similar ao sscanf de C, usamos um texto como um arquivo.
        stream_string = std::istringstream(linha);
        std::string nome;
        int matricula;
        int codigo_disciplina;
        stream_string >> nome;
        stream_string >> matricula;
        stream_string >> codigo_disciplina;

        Aluno aluno(nome, matricula);
    }
}
```

A alocação de memória para o objeto `Aluno` é considerada inútil no momento.

# Execução

Exemplo de como o programa executa

```
$ ./main
Flavio 2018101 01
Flavio 2018101 02
Flavio 2018101 04
Ana 2018103 03
Ana 2018103 02
sair
```

# Streams e Arquivos

São quase iguais

- Por enquanto usamos stream cin
- Para mudar para um arquivo é simples
  - Ver fim dos slides

# Problema 01

- Dada uma classe Aluno
- Data um arquivo de Nome, Aluno, Matéria
- Como:
  - Alocar um novo aluno para cada registro (linha)
  - Pegar os nomes únicos dos alunos

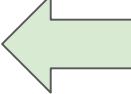
# Problema 01

## Qual o melhor TAD?

- Lista
- Vector
- Set
- Map
- Bignum

# Problema 01

## Qual o melhor TAD?

- Lista**
- Vector**
- Set** 
- Map**
- Bignum**

# Problema 01

## Lembrando de Sets

- Guardam elementos únicos
- Métodos `insert` e `remove`

```
// . . .
#include <set>
// . . .

int main() {
    std::string linha;
    std::istringstream stream_string;
    std::set<std::string> nomes;

    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        stream_string = std::istringstream(linha);
        std::string nome;
        int matricula;
        int codigo_disciplina;
        stream_string >> nome;
        stream_string >> matricula;
        stream_string >> codigo_disciplina;

        Aluno aluno(nome, matricula);
        nomes.insert(aluno.get_nome());
    }
    // . . .
}
```

```
// . . .
#include <set>
// . . .

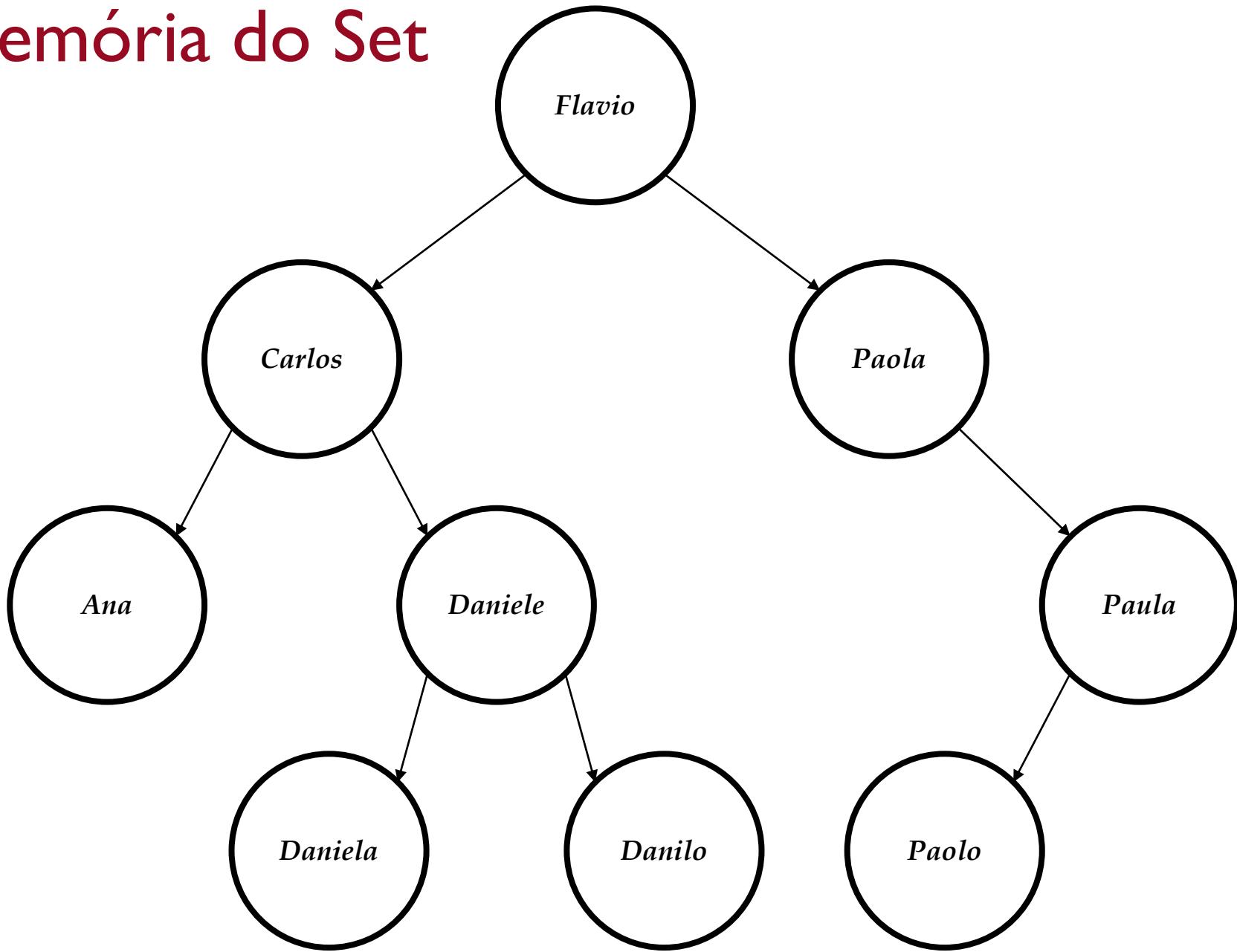
int main() {
    std::string linha;
    std::istringstream stream_string;
    std::set<std::string> nomes;

    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        stream_string = std::istringstream(linha);
        std::string nome;
        int matricula;
        int codigo_disciplina;
        stream_string >> nome;
        stream_string >> matricula;
        stream_string >> codigo_disciplina;

        Aluno aluno(nome, matricula);
        nomes.insert(aluno.get_nome());
    }
    // . . .
}
```

Inserção em uma árvore

# Memória do Set



# Solução

Usando a entrada padrão (`std::cin` ou teclado)

```
// . . .
#include <set>
// . . .
int main() {
    // . . .
    while (std::getline(std::cin, linha)) {
        // . . .
        Aluno aluno(nome, matricula);
        nomes.insert(aluno.get_nome());
    }
    for (std::string nome : nomes) {
        std::cout << nome << std::endl;
    }
}
```

Foreach para impressão

# Execução

Exemplo de como o programa executa

```
$ ./main
Flavio 2018101 01
Flavio 2018101 02
Flavio 2018101 04
Ana 2018103 03
Ana 2018103 02
sair
```

```
Ana
Flavio
```

# Problema 01a

- Dada uma classe Aluno
- Data um arquivo de Nome, Aluno, Matéria
- Como:
  - Alocar um novo aluno para cada registro (linha)
  - Representar os alunos em um set por matrícula

# Usando classes em Containers

- Precisamos saber como comparar o Objeto
- Strings são comparadas por ordem lexicográfica
- Números por ordem natural (<)
- Como fazemos para as outras classes?

# Functor Comparator

- Comparator é um termo comum para:
  - Operação com único propósito é apenas comparar dois objetos
  - Podemos colocar dentro de um struct
- Functor
  - TAD com que tem um operador
  - Este operador pode comparar dois TADs

# Functor Comparator

```
// Functor comparator com struct
struct aluno_comparator_f {
    bool operator()(const Aluno &aluno1, const Aluno &aluno2) const {
        return aluno1.get_matricula() < aluno2.get_matricula();
    }
};
```

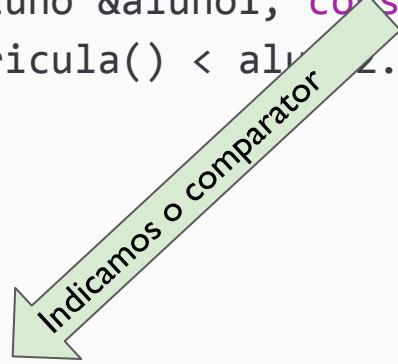
Precisa do const, não altera mem

Precisa do const, não altera mem

- Note o método `operator`
- O mesmo é chamado internamente para comparar dois alunos no set/map

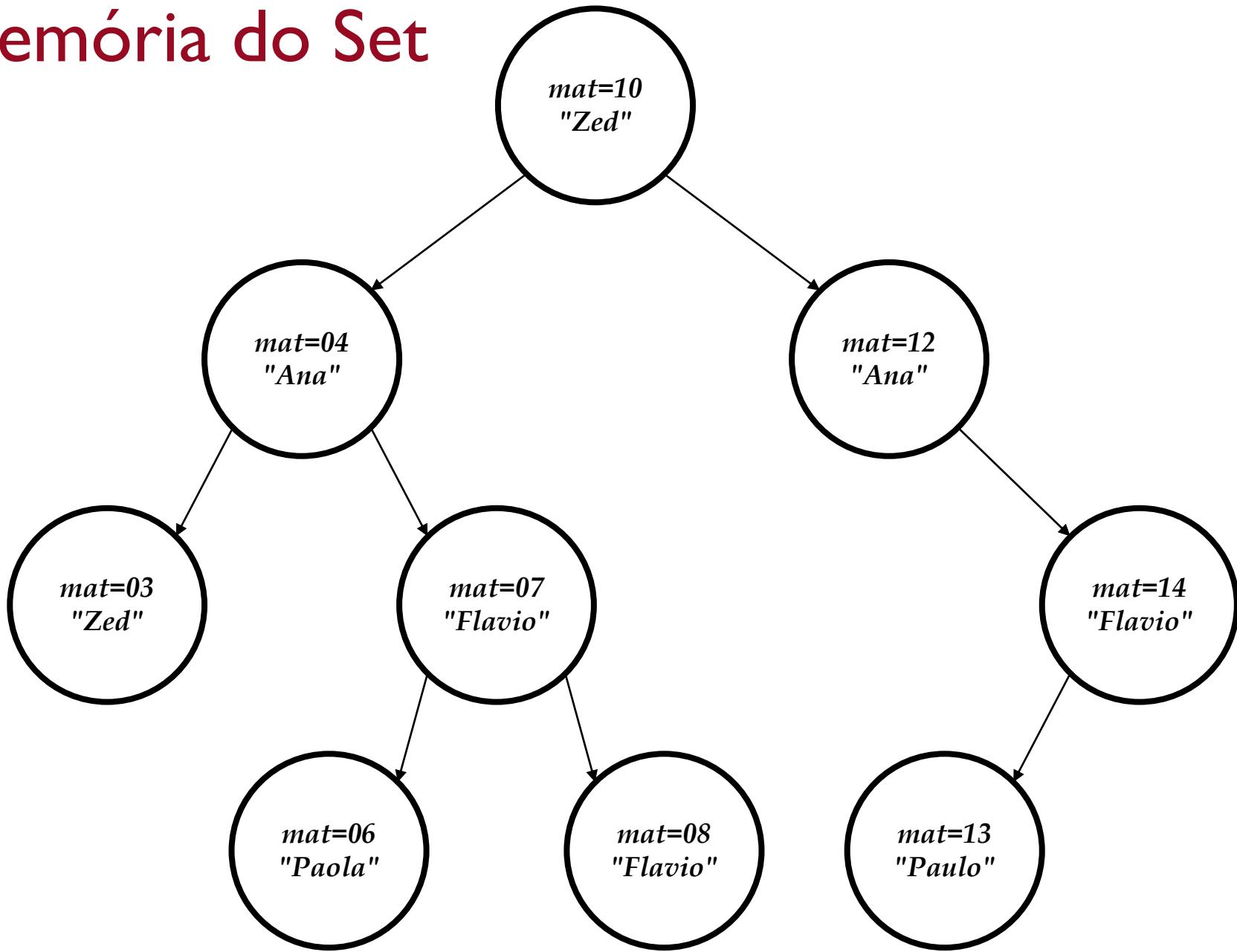
```
// . . .  
  
// Functor comparator  
struct aluno_comparator_f {  
    bool operator()(const Aluno &aluno1, const Aluno &aluno2) const {  
        return aluno1.get_matricula() < aluno2.get_matricula();  
    }  
};  
  
int main() {  
    // . . .  
    std::set<Aluno, aluno_comparator_f> nomes;  
    while (std::getline(std::cin, linha)) {  
        // . . .  
        Aluno aluno(nome, matricula);  
        nomes.insert(aluno);  
    }  
    for (Aluno aluno : nomes) {  
        std::cout << aluno.get_nome() << std::endl;  
    }  
}
```

```
// . . .  
  
// Functor comparator  
struct aluno_comparator_f {  
    bool operator()(const Aluno &aluno1, const Aluno &aluno2) const {  
        return aluno1.get_matricula() < aluno2.get_matricula();  
    }  
};  
  
int main() {  
    // . . .  
    std::set<Aluno, aluno_comparator_f> nomes;  
    while (std::getline(std::cin, linha)) {  
        // . . .  
        Aluno aluno(nome, matricula);  
        nomes.insert(aluno);  
    }  
    for (Aluno aluno : nomes) {  
        std::cout << aluno.get_nome() << std::endl;  
    }  
}
```

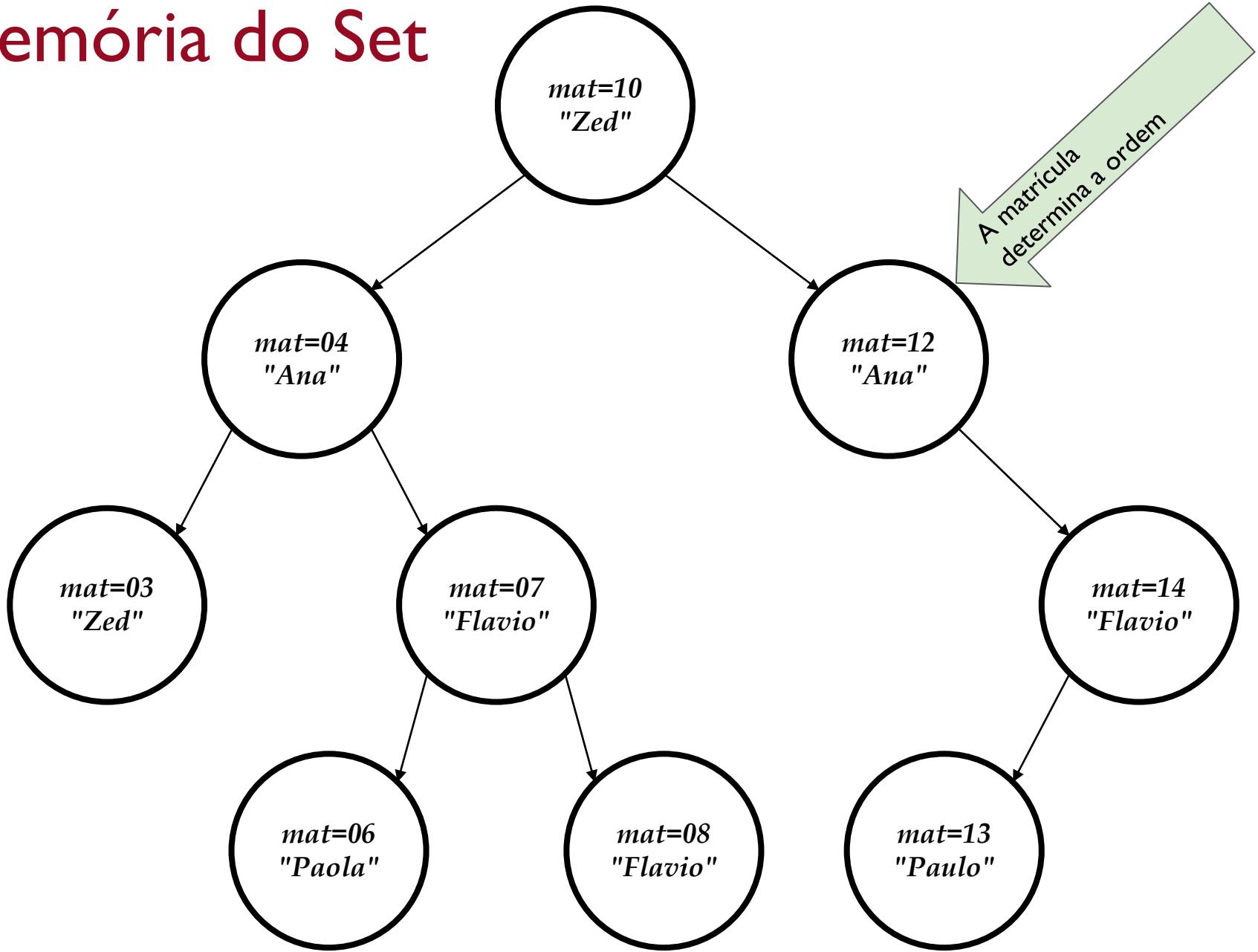


Indicamos o comparador

# Memória do Set



# Memória do Set



# Execução

Exemplo de como o programa executa

```
$ ./main
Flavio 2018101 01
Flavio 2018101 02
Flavio 2018102 01
Ana 2018103 03
Ana 2018103 02
sair
```

```
Flavio
Flavio
Ana
```

# Problema 02

- Dado um arquivo grande de texto. Em cada linha, temos um texto diferente (p. ex., tweets, documentos, conversas etc.).
- Como:
  - Carregar o arquivo inteiro na memória do computador?
  - Mantendo a ordem

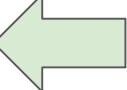
# Problema 02

## Qual o melhor TAD?

- Lista
- Vector
- Set
- Map
- Bignum

# Problema 02

## Qual o melhor TAD?

- Lista
- Vector 
- Set
- Map
- Bignum

# Solução

```
// . . .
#include <vector>

int main() {
    std::string linha;
    std::string palavra;

    std::vector<std::string> linhas;

    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        linhas.push_back(linha);
    }
    // . . .
}
```

# Problema 03

- Dado um arquivo grande de texto. Em cada linha, temos um texto diferente (p. ex., tweets, documentos, conversas etc.).
- Como:
  - Carregar o arquivo na memória do computador?
  - Filtrar as linhas que aparecem algum termo?

# Problema 03 (a) e (b)

- Dado um arquivo grande de texto. Em cada linha, temos um texto diferente (p. ex., tweets, documentos, conversas etc.).
- Como:
  - Carregar o arquivo na memória do computador?
  - Filtrar as linhas que aparecem algum termo?
    - (a) In-place: Com remoção
    - (b) Criando um novo objeto

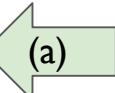
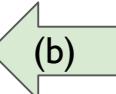
# Problema 03

## Qual o melhor TAD?

- Lista
- Vector
- Set
- Map
- Bignum

# Problema 03

## Qual o melhor TAD?

- Lista** 
- Vector** 
- Set**
- Map**
- Bignum**

# Solução (a)

```
int main() {
    // ...
    std::list<std::string> linhas;
    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        linhas.push_back(linha);
    }

    std::string termo;
    std::cout << "Digite o termo para filtrar" << std::endl;
    std::cin >> termo;

    std::list<std::string>::iterator it = linhas.begin();
    while (it != linhas.end()) {
        linha = *it;
        int posicao = linha.find(termo);
        if (posicao != std::string::npos)
            linhas.erase(it);
        it++;
    }

    for (std::string linha : linhas) {
        std::cout << linha << std::endl;
    }
}
```

# Solução (a)

```
int main() {
    // ...
    std::list<std::string> linhas;
    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        linhas.push_back(linha);
    }

    std::string termo;
    std::cout << "Digite o termo para filtrar" << std::endl;
    std::cin >> termo;

    std::list<std::string>::iterator it = linhas.begin(); begin é o inicio
    while (it != linhas.end()) { end é o fim
        linha = *it;
        int posicao = linha.find(termo);
        if (posicao != std::string::npos)
            linhas.erase(it);
        it++;
    }

    for (std::string linha : linhas) {
        std::cout << linha << std::endl;
    }
}
```

# Solução (a)

```
int main() {
    // ...
    std::list<std::string> linhas;
    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        linhas.push_back(linha);
    }

    std::string termo;
    std::cout << "Digite o termo para filtrar" << std::endl;
    std::cin >> termo;

    std::list<std::string>::iterator it = linhas.begin(); begin é o inicio
    while (it != linhas.end()) { end é o fim
        linha = *it;
        int posicao = linha.find(termo);
        if (posicao != std::string::npos)
            linhas.erase(it);
        it++; lembrando que iteradores funcionam parecido com ponteiros
    }

    for (std::string linha : linhas) {
        std::cout << linha << std::endl;
    }
}
```

# Solução (a)

```
int main() {
    // ...
    std::list<std::string> linhas;
    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        linhas.push_back(linha);
    }

    std::string termo;
    std::cout << "Digite o termo para filtrar" << std::endl;
    std::cin >> termo;

    std::list<std::string>::iterator it = linhas.begin();
    while (it != linhas.end()) {
        linha = *it;
        int posicao = linha.find(termo);
        if (posicao != std::string::npos)
            linhas.erase(it); ← Apaga o elemento atual!
        it++;
    }

    for (std::string linha : linhas) {
        std::cout << linha << std::endl;
    }
}
```

## Solução (a)



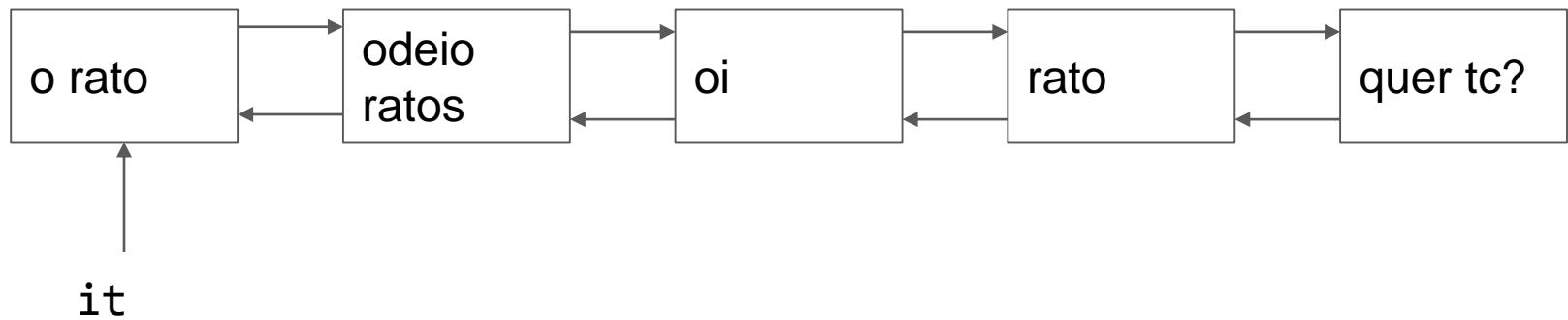
```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```



it

## Solução (a)

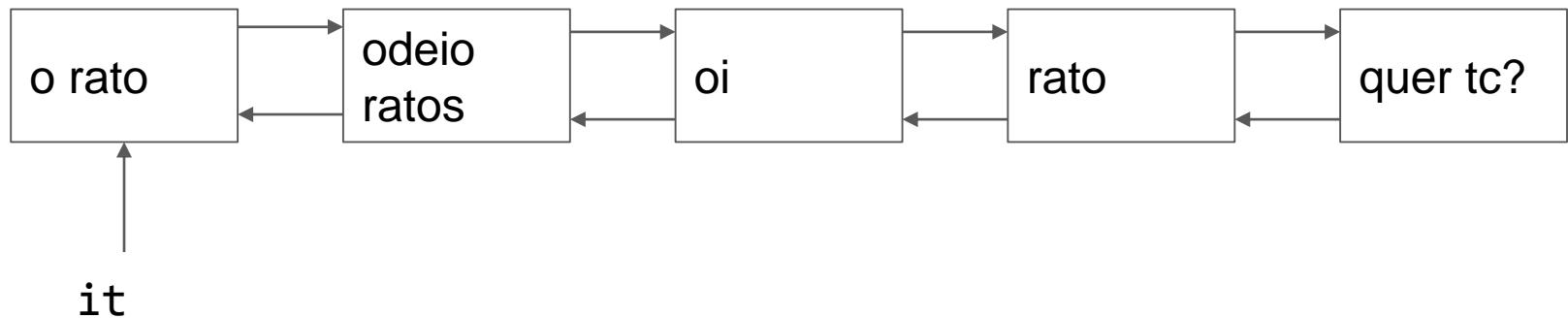
```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```



termo = rato

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```

## Solução (a)



termo = rato

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```

## Solução (a)



it  
linha = "o rato";

termo = rato

```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

## Solução (a)



it  
 linha = "o rato";  
 posicao = 2; // indica onde em "o rato" existe "rato"  
 // [0] = 'o'; [1] = ' '; [2] = 'r'; ...

termo = rato

```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

## Solução (a)



it

```

linha = "o rato";
posicao = 2;
std::string::npos indica -1, não achou

```

termo = rato

```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

# Solução (a)

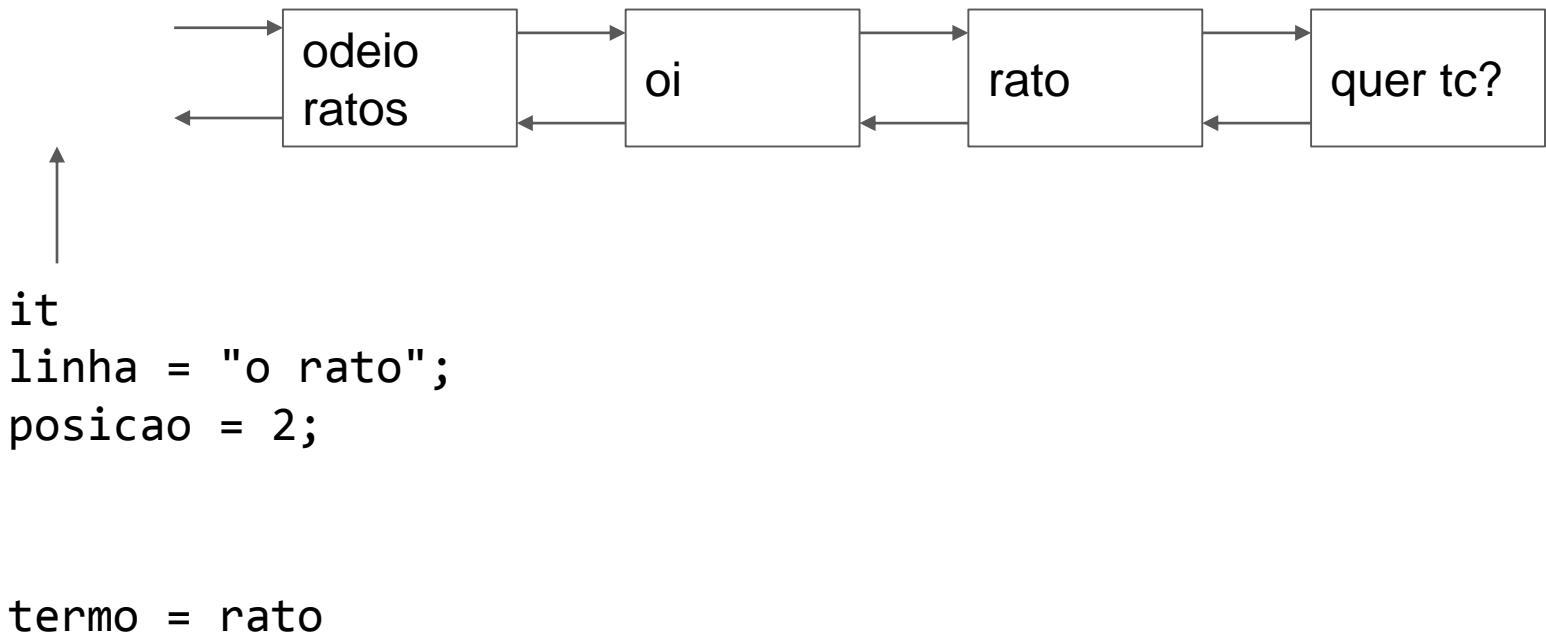


it  
 linha = "o rato";  
 posicao = 2;

termo = rato

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    →it++;
}
```

## Solução (a)

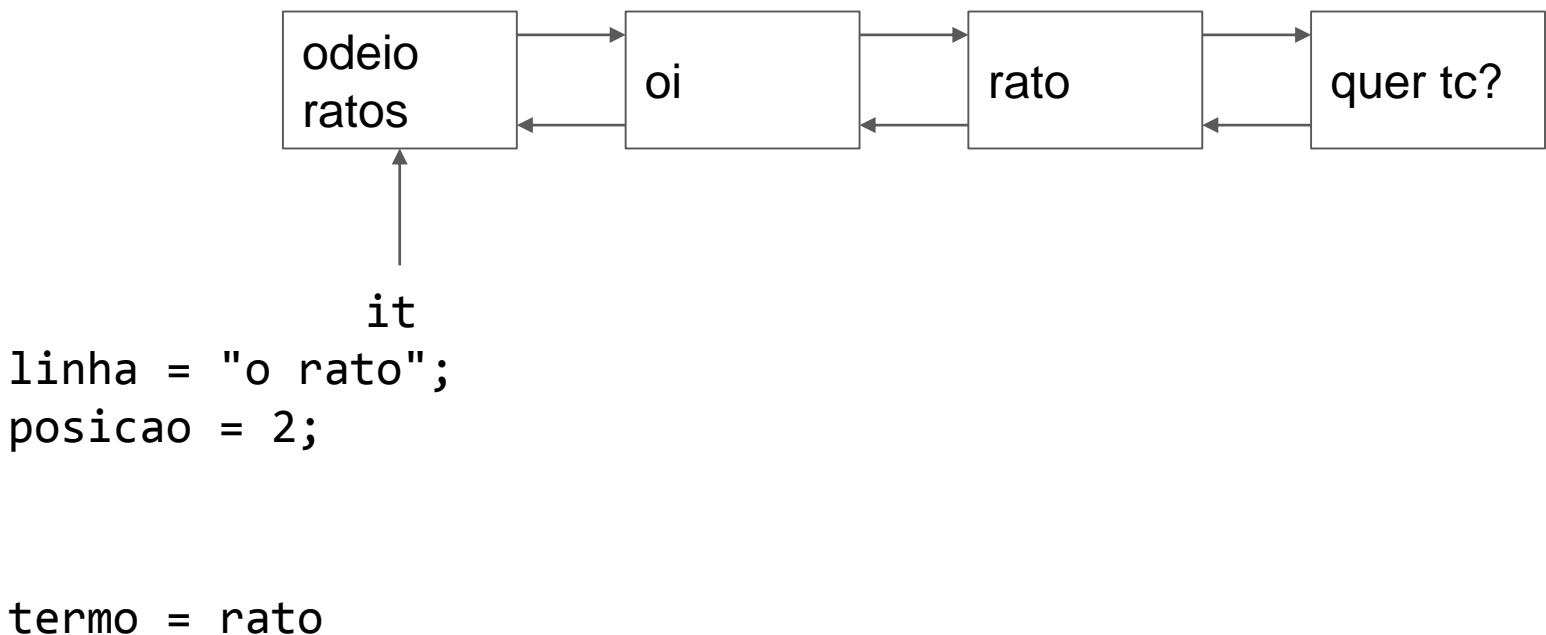


```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

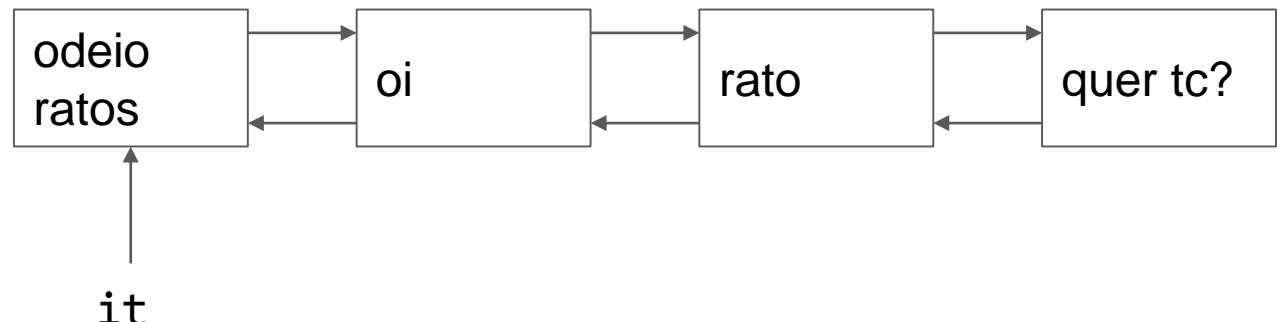
```

# Solução (a)



```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```

## Solução (a)

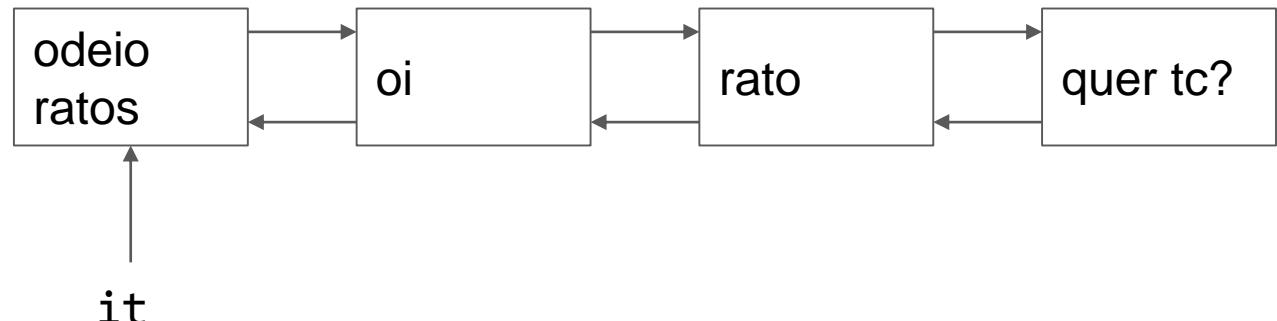


linha = "odeio ratos";  
posicao = 2;

termo = rato

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```

## Solução (a)



```
linha = "odeio ratos";
posicao = 6;
```

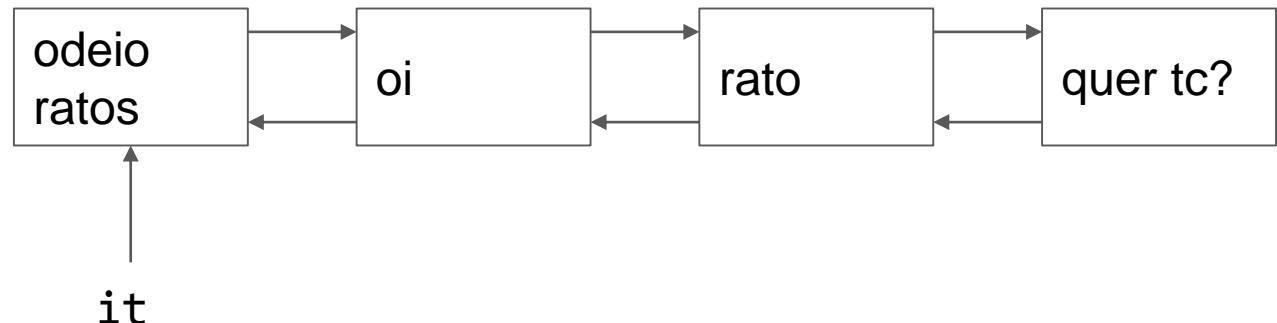
termo = rato

```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

# Solução (a)



linha = "odeio ratos";  
 posicao = 6;

termo = rato

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```

## Solução (a)

linha = "odeio ratos";  
posicao = 6;

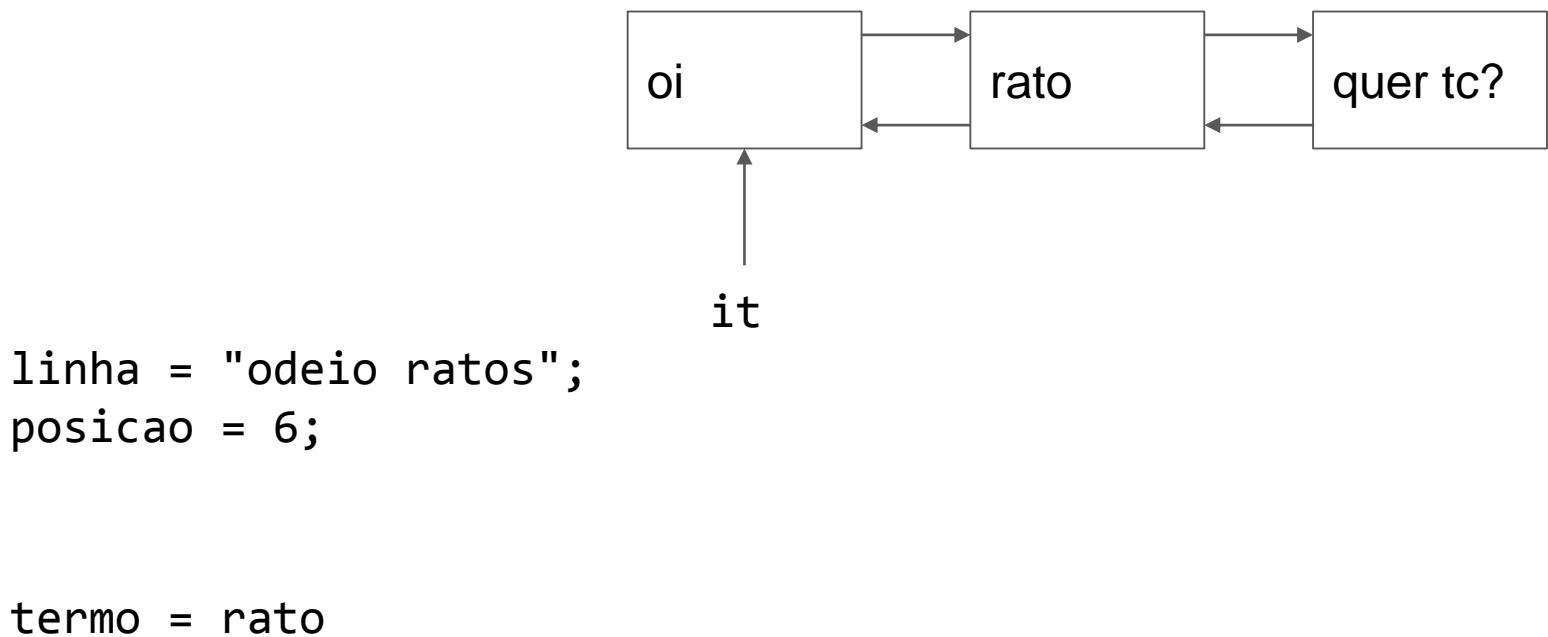
termo = rato

↑  
it



# Solução (a)

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```



```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

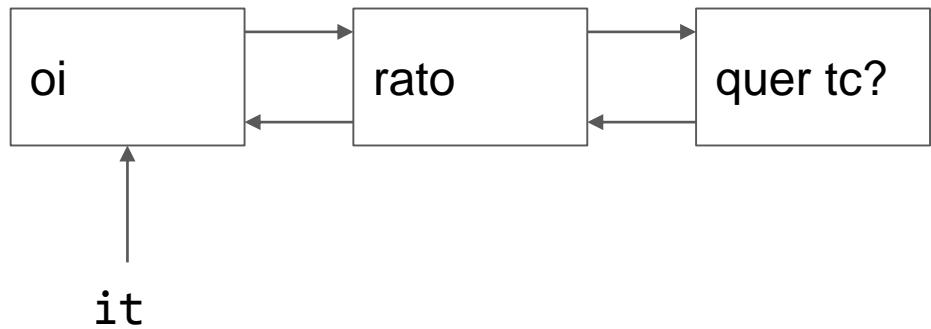
# Solução (a)

```

linha = "oi";
posicao = -1;

```

termo = rato

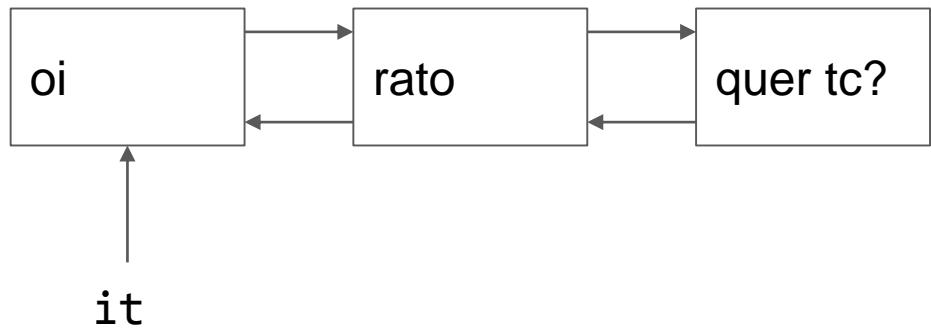


```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    →it++;
}
```

## Solução (a)

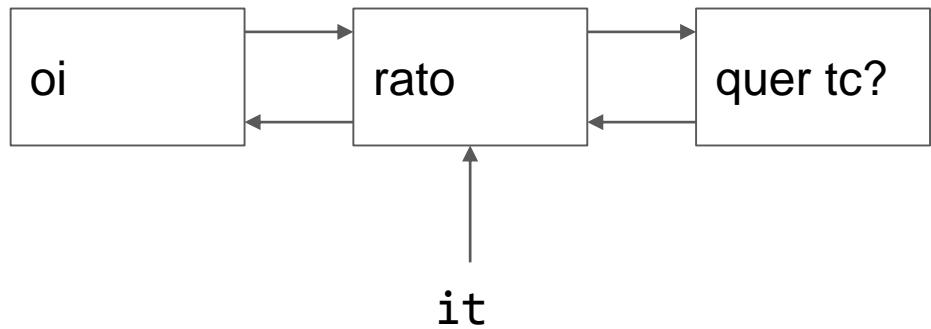
```
linha = "oi";
posicao = -1;
```

termo = rato



# Solução (a)

```
std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}
```



```
linha = "oi";
posicao = -1;
```

termo = rato

```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

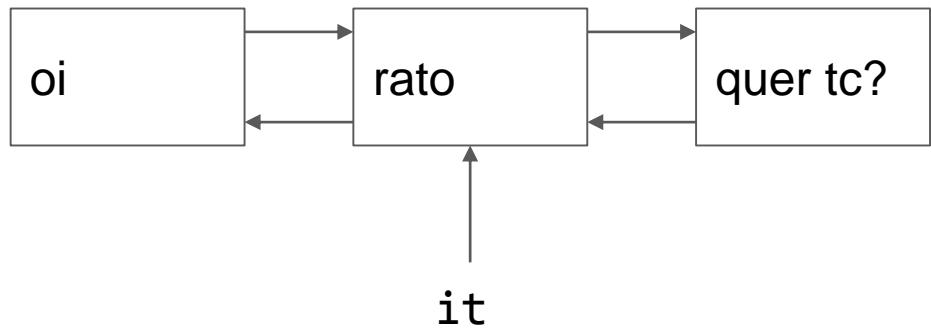
# Solução (a)

```

linha = "oi";
posicao = -1;

```

termo = rato



```

std::list<std::string>::iterator it = linhas.begin();
while (it != linhas.end()) {
    linha = *it;
    int posicao = linha.find(termo);
    if (posicao != std::string::npos)
        linhas.erase(it);
    it++;
}

```

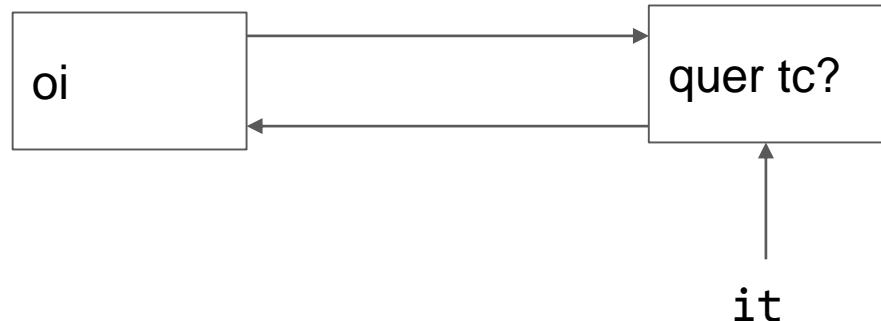
## Solução (a)

```

linha = "oi";
posicao = -1;

```

termo = rato



# Solução (b)

- Note o uso de referências
- Retornamos um novo vetor
- Operando em cima da referência

```
std::vector<std::string> filtrar(const std::vector<std::string> &linhas, std::string termo) {  
    std::vector<std::string> resposta;  
    for (std::string linha : linhas) {  
        if (linha.find(termo) == std::string::npos) {  
            resposta.push_back(linha);  
        }  
    }  
    return resposta;  
}
```

# Problema 04

- Dado um arquivo grande de texto. Em cada linha, temos um texto diferente (p. ex., tweets, documentos, conversas etc.).
- Como:
  - Carregar o arquivo na memória do computador?
  - Filtrar as linhas que aparecem algum termo?
  - Contar quantas vezes cada palavra aparece?

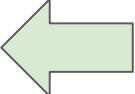
# Problema 04

## Qual o melhor TAD?

- Lista
- Vector
- Set
- Map
- Bignum

# Problema 04

## Qual o melhor TAD?

- Lista
- Vector
- Set
- Map 
- Bignum

# Solução

```
int main() {
    std::string linha;
    std::istringstream stream_string;

    std::map<std::string, int> contagem;
    std::string termo;
    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        stream_string = std::istringstream(linha);
        while (std::getline(stream_string, termo, ' ')) {
            if (contagem.count(termo) == 0) { // 0 se não existe; 1 cc.
                contagem[termo] = 0;
            }
            contagem[termo] = contagem[termo] + 1;
        }
    }

    std::cout << std::endl;
    for (std::pair<std::string, int> pair : contagem) {
        std::cout << "Termo : " << pair.first << " Contagem: " << pair.second;
        std::cout << std::endl;
    }
}
```

# Solução

```
int main() {
    std::string linha;
    std::istringstream stream_string;

    std::map<std::string, int> contagem; // Mapa de string → inteiros
    std::string termo;
    while (std::getline(std::cin, linha)) {
        if (linha == "sair")
            break;
        stream_string = std::istringstream(linha);
        while (std::getline(stream_string, termo, ' ')) { // Similar o get line, separando por ' '
            if (contagem.count(termo) == 0) { // Testa se existe
                contagem[termo] = 0;
            }
            contagem[termo] = contagem[termo] + 1;
        }
    }

    std::cout << std::endl;
    for (std::pair<std::string, int> pair : contagem) {
        std::cout << "Termo : " << pair.first << " Contagem: " << pair.second;
        std::cout << std::endl;
    }
}
```

# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```



```
linha = "tudo bem oi tudo bem";
termo = tudo;
```

# Passo a Passo

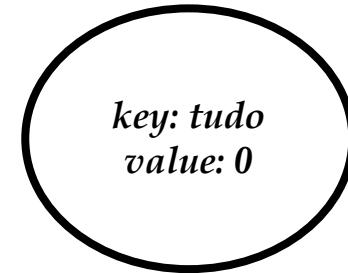
```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        → contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```

```
linha = "tudo bem oi tudo bem";
termo = tudo;
```

# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    → contagem[termo] = contagem[termo] + 1;
}
```

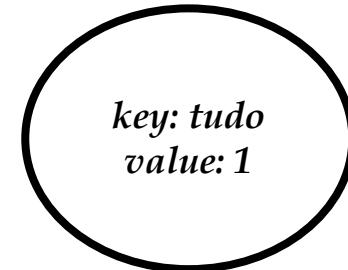
```
linha = "tudo bem oi tudo bem";
termo = tudo;
```



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```

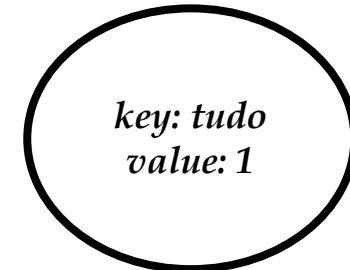
```
linha = "tudo bem oi tudo bem";
termo = bem;
```



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        → contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```

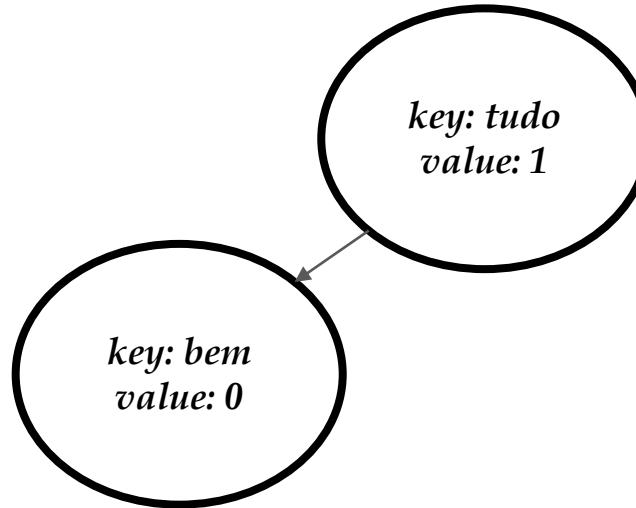
```
linha = "tudo bem oi tudo bem";
termo = bem;
```



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    → contagem[termo] = contagem[termo] + 1;
}
```

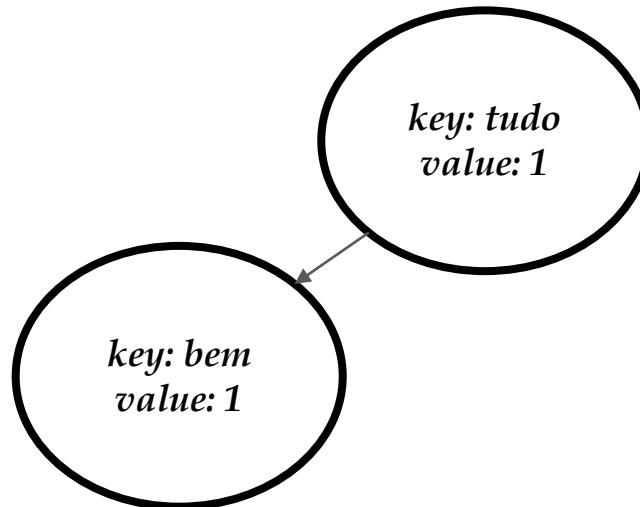
```
linha = "tudo bem oi tudo bem";
termo = bem;
```



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```

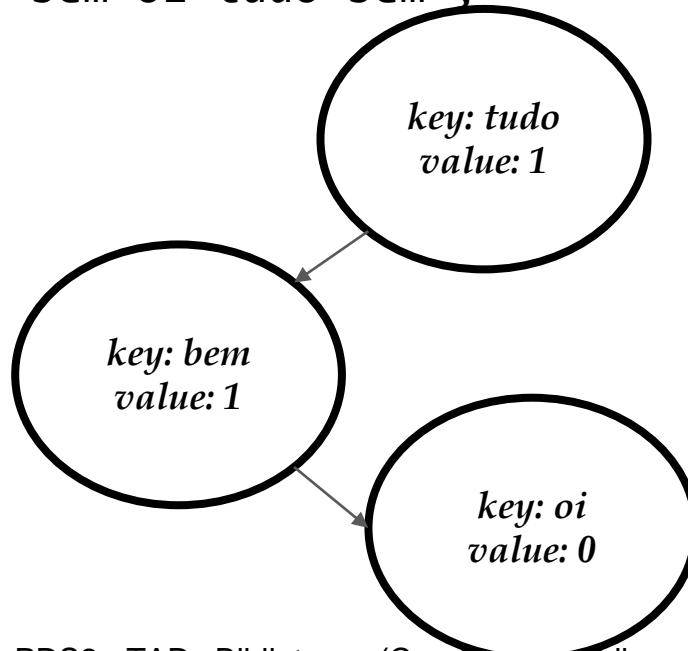
```
linha = "tudo bem oi tudo bem";
termo = oi;
```



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    → contagem[termo] = contagem[termo] + 1;
}
```

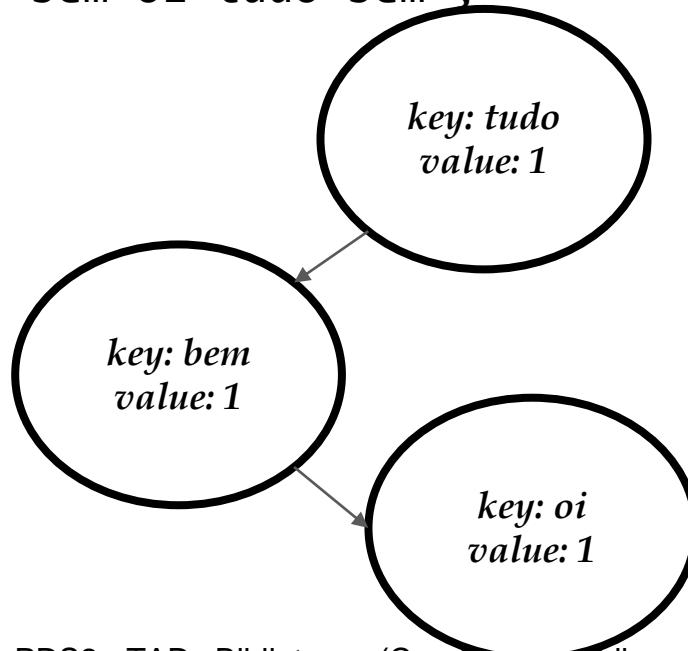
linha = "tudo bem oi tudo bem";  
termo = oi;



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```

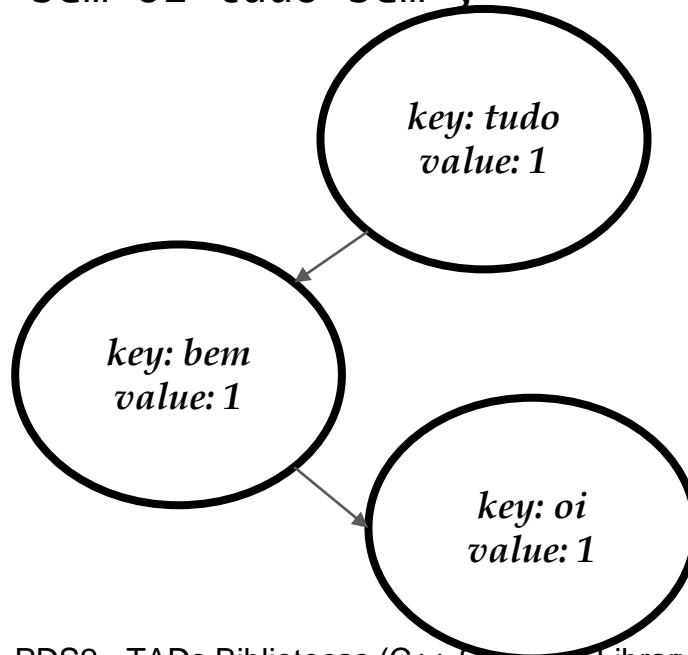
linha = "tudo bem oi tudo bem";  
termo = tudo;



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    → contagem[termo] = contagem[termo] + 1;
}
```

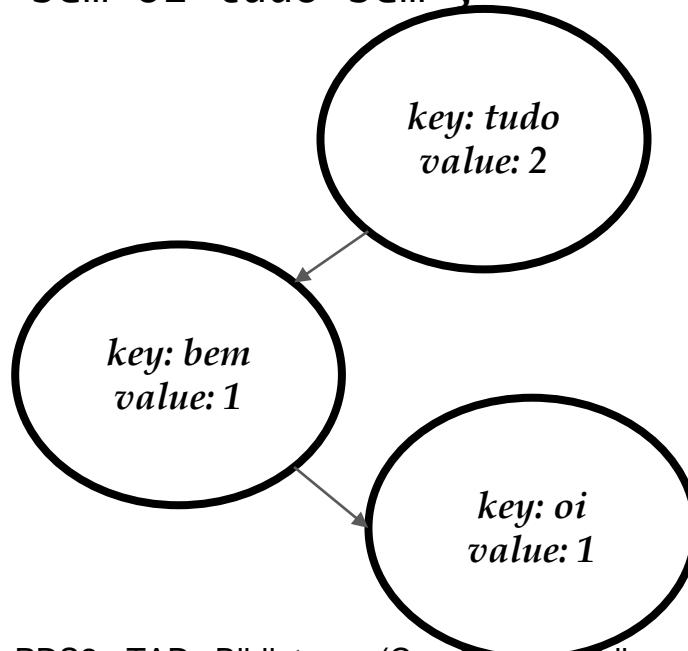
linha = "tudo bem oi tudo bem";  
termo = tudo;



# Passo a Passo

```
stream_string = std::istringstream(linha);
while (std::getline(stream_string, termo, ' ')) {
    if (contagem.count(termo) == 0) {
        contagem[termo] = 0;
    }
    contagem[termo] = contagem[termo] + 1;
}
```

linha = "tudo bem oi tudo bem";  
termo = bem;



# Usando Arquivos

Três pequenas mudanças, resto continua o mesmo

```
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string>

int main() {
    std::ifstream entrada("nome_do_arquivo.txt");
    std::string linha;
    while (std::getline(entrada, linha)) {
        // . . .
    }
    // . . .
    entrada.close();
}
```

# Usando Arquivos

Três pequenas mudanças, resto continua o mesmo

```
#include <fstream>
#include <iostream>
#include <map>
#include <sstream>
#include <string>

int main() {
    std::ifstream entrada("nome_do_arquivo.txt");
    std::string linha;
    while (std::getline(entrada, linha)) {
        // ...
    }
    // ...
    entrada.close();
}
```

Abre arquivo

Usamos o arquivo e não cin

Fecha

# Problema 05

- Dada uma classe Aluno
- Data um arquivo de Nome, Aluno, Matéria
- Como:
  - Alocar um novo aluno para cada registro (linha)
  - Pegar os nomes únicos dos alunos
  - Contar quantas matérias cada aluno está matriculado

# Problema 05

Solução 01:

Guardar um contador na classe

Solução 02:

Guardar um set na classe

Solução 03:

Fazer um mapa contador

# Problema 05

- Dada uma classe Aluno
- Data um arquivo de Nome, Aluno, Matéria
- Como:
  - Alocar um novo aluno para cada registro (linha)
  - Pegar os nomes únicos dos alunos
  - Contar quantas matérias cada aluno está matriculado

# Problema 06

- Dado um arquivo grande de texto. Em cada linha, temos um texto diferente (p. ex., tweets, documentos, conversas etc.).
- Como criar um índice invertido?
  - ? map<std::string, std::set<int>>
  - Cada entrada do mapa indica as linhas que um termo ocorre.

# Problema 06

## Arquivo

- 1: o rato roeu a roupa do rei de roma
- 2: roma é uma cidade
- 3: bh é uma cidade

# Problema 06

## Arquivo

- 1: o rato roeu a roupa do rei de roma
- 2: roma é uma cidade
- 3: bh é uma cidade

## Mapa

rato → {1}

roma → {1, 2}

cidade → {2, 3}

• • •

# Programação e Desenvolvimento de Software 2

## Programação Orientada a Objetos

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# Introdução

- Programação Estruturada
  - Instruções que mudam o estado do programa
  - Programas imperativos (ações)
- Programação Orientada a Objetos
  - Dados e procedimentos encapsulados
  - Composto por diversos objetos
  - Interação/comunicação entre os objetos

# Introdução

## Programação Estruturada

- Como resolver problemas muito grandes?
  - Construí-lo a partir de partes menores
- Módulos compiláveis
  - Solucionam uma parte do problema
  - Dados x Manipulação
    - Abstração fraca para problemas mais complexos

# Programação Orientada a Objetos

- Sistemas maiores e mais complexos
  - Aumentar a produtividade no desenvolvimento
  - Diminuir a chance de problemas
  - Facilitar a manutenção/extensão
- Programação Orientada a Objetos
  - Tem apresentado bons resultados
  - Não é uma bala de prata!

[https://en.wikipedia.org/wiki/Object-oriented\\_programming#Criticism](https://en.wikipedia.org/wiki/Object-oriented_programming#Criticism)

# Programação Orientada a Objetos

## História

- Desenvolvimento de hardware
  - Pedaços simples de hardware (chips) unidos para se montar um hardware mais complexo
- Amadurecimento dos conceitos
  - Simula (60's)
  - Smalltalk (70's)
  - C++ (80's)

# Programação Orientada a Objetos

## PE vs. POO

- Programação Estruturada
  - Procedimentos implementados em blocos
  - Comunicação pela passagem de dados
  - Execução → Açãoamento de procedimentos
- Programação Orientada a Objetos
  - Dados e procedimentos encapsulados
  - Execução → Comunicação entre objetos

# Programação Orientada a Objetos

Novo paradigma de programação

- Programação Estruturada
  - Dados acessados via funções
  - Representação de tipos complexos com struct
- Programação Orientada a Objetos
  - Dados são dotados de certa inteligência
  - Sabem realizar operações sobre si mesmos
  - É preciso conhecer a implementação?

# Programação Orientada a Objetos

## Benefícios

- Maior confiabilidade
- Maior reaproveitamento de código
- Facilidade de manutenção
- Melhor gerenciamento
- Maior robustez
- ...

# Classes vs Objetos vs TADs

## □ Classe/Struct

- Representa uma unidade de compilação
- Um módulo, um tipo
- Pode implementar um TAD
- Em C++, classes e structs são quase iguais

## □ TAD

- Conceito, ideia, abstração (representação)

# Classes vs Objetos

- Classes representam a **forma** da memória
- Objetos são **instâncias** de classes

# Analogia

- Classes → fôrmas
- Memória → massa
- Objetos → cookies

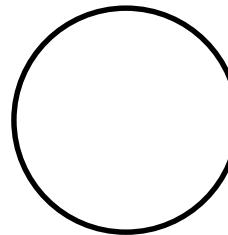


# Classes vs. Objetos

- **Classe**
  - Descrição de propriedades em comum de um grupo de objetos (conjunto)
  - Um conceito
  - Faz parte de um programa
  - Exemplo: Pessoa
  - Exemplo: Carro
- **Objeto**
  - Representação das propriedades de uma única instância (elemento)
  - Um fenômeno (ocorrência)
  - Faz parte de uma execução
  - Exemplo: João da Silva
  - Exemplo: Ferrari

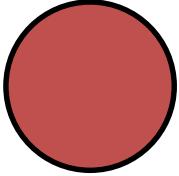
# Classes vs. Objetos

CLASSE

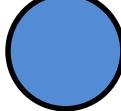


Peso  
Raio  
Cor

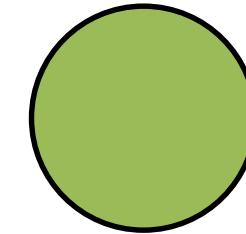
OBJETOS



Peso: 100 g  
Raio: 25 cm  
Cor: Vermelha



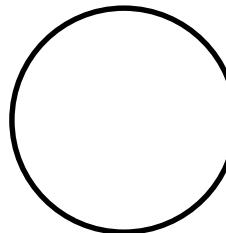
Peso: 50 g  
Raio: 10 cm  
Cor: Azul



Peso: 200 g  
Raio: 30 cm  
Cor: Verde

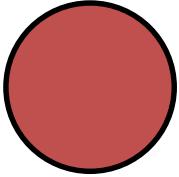
# Classes vs. Objetos

CLASSE

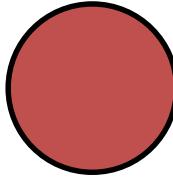


Peso  
Raio  
Cor

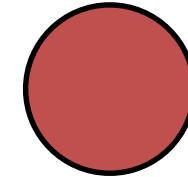
OBJETOS



Peso: 100 g  
Raio: 25 cm  
Cor: Vermelha



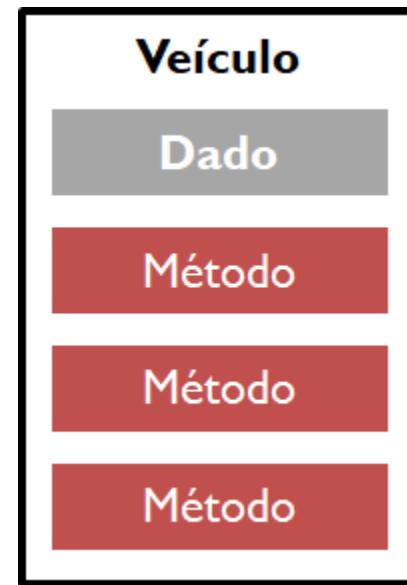
Peso: 100 g  
Raio: 25 cm  
Cor: Vermelha



Peso: 100 g  
Raio: 25 cm  
Cor: Vermelha

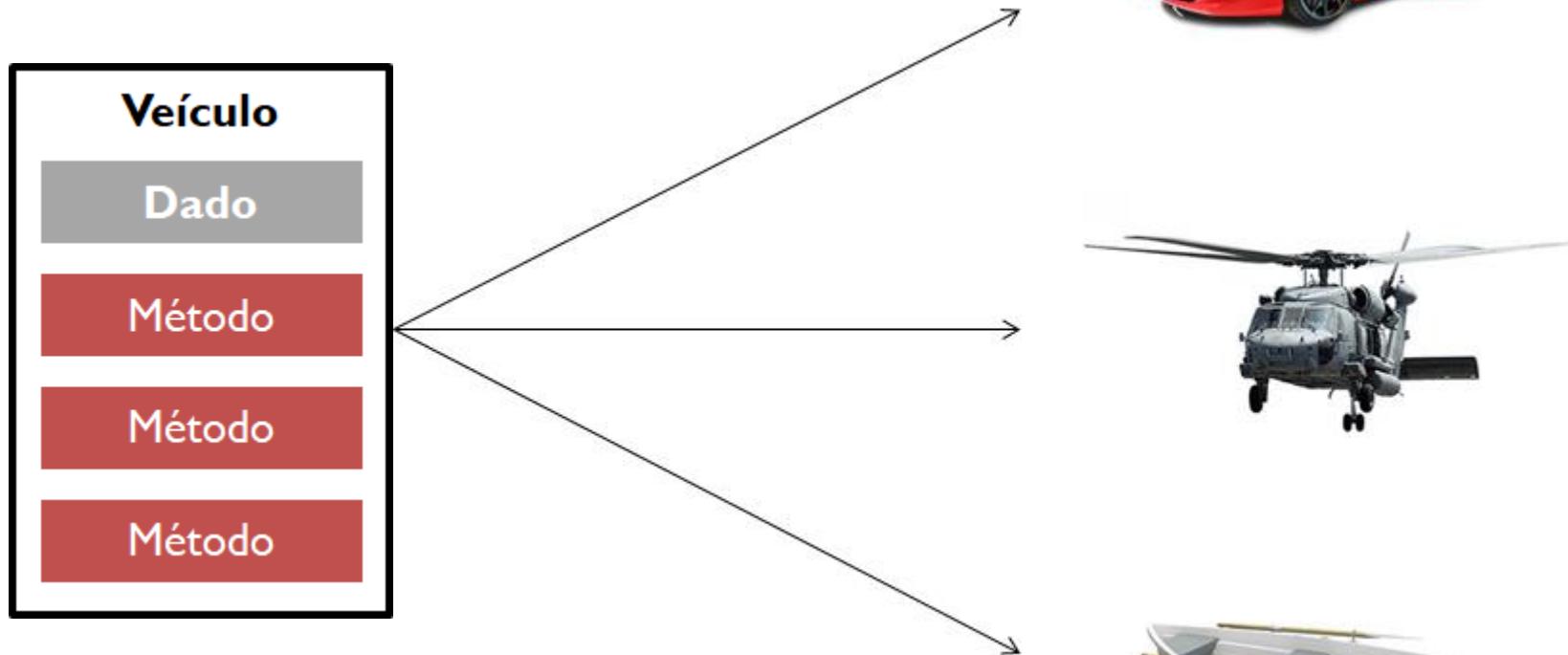
# Classes

## Abstração



# Classes

## Abstração



# Exemplo de Classe

```
class Ponto {  
private:  
    float _x;  
    float _y;  
public:  
    Ponto(float x, float y) {  
        _x = x;  
        _y = y;  
    }  
    float get_x() {  
        return _x;  
    }  
    float get_y() {  
        return _y;  
    }  
    void translacao(double dx, double dy) {  
        _x += dx;  
        _y += dy;  
    }  
};
```

# Exemplo de Classe

```
class Ponto {  
private:  
    float _x;    ← Atributos da classe  
    float _y;  
public:  
    Ponto(float x, float y) { ← Construtor  
        _x = x;  
        _y = y;  
    }  
    float get_x() { ← Getters  
        return _x;  
    }  
    float get_y() {  
        return _y;  
    }  
    void translacao(double dx, double dy) { ← Nossa função de translação  
        _x += dx;  
        _y += dy;  
    }  
};
```

# Usando o objeto

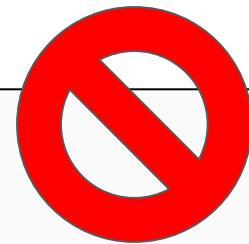
- Para utilizar o objeto usamos os métodos
  - Funções

```
#include <iostream>
// . . . Declaracao do Ponto aqui em cima . . . //
int main() {
    Ponto ponto(7, 9);
    std::cout << "Valor de x: " << ponto.get_x() << std::endl;
    std::cout << "Valor de y: " << ponto.get_y() << std::endl;
    ponto.translacao(3, 1);
    std::cout << "Valor de x: " << ponto.get_x() << std::endl;
    std::cout << "Valor de y: " << ponto.get_y() << std::endl;
}
```

# Usando o objeto

- Não temos acesso direto aos atributos
- Erro de compilação

```
#include <iostream>
// . . . Declaracao do Ponto aqui em cima . . . //
int main() {
    Ponto ponto(7, 9);
    std::cout << "Valor de x: " << ponto._x << std::endl;
    std::cout << "Valor de y: " << ponto._y << std::endl;
}
```

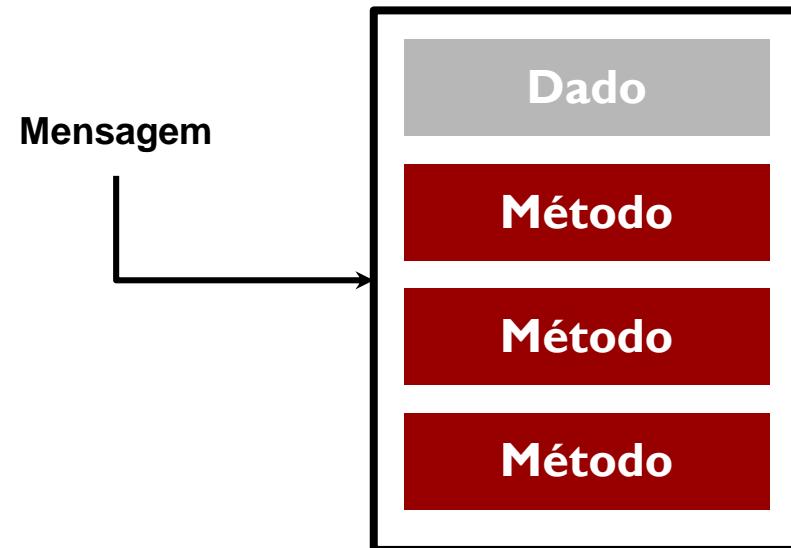


# Objetos

- Dados ocultos do “mundo externo”
- Acessíveis somente via métodos internos

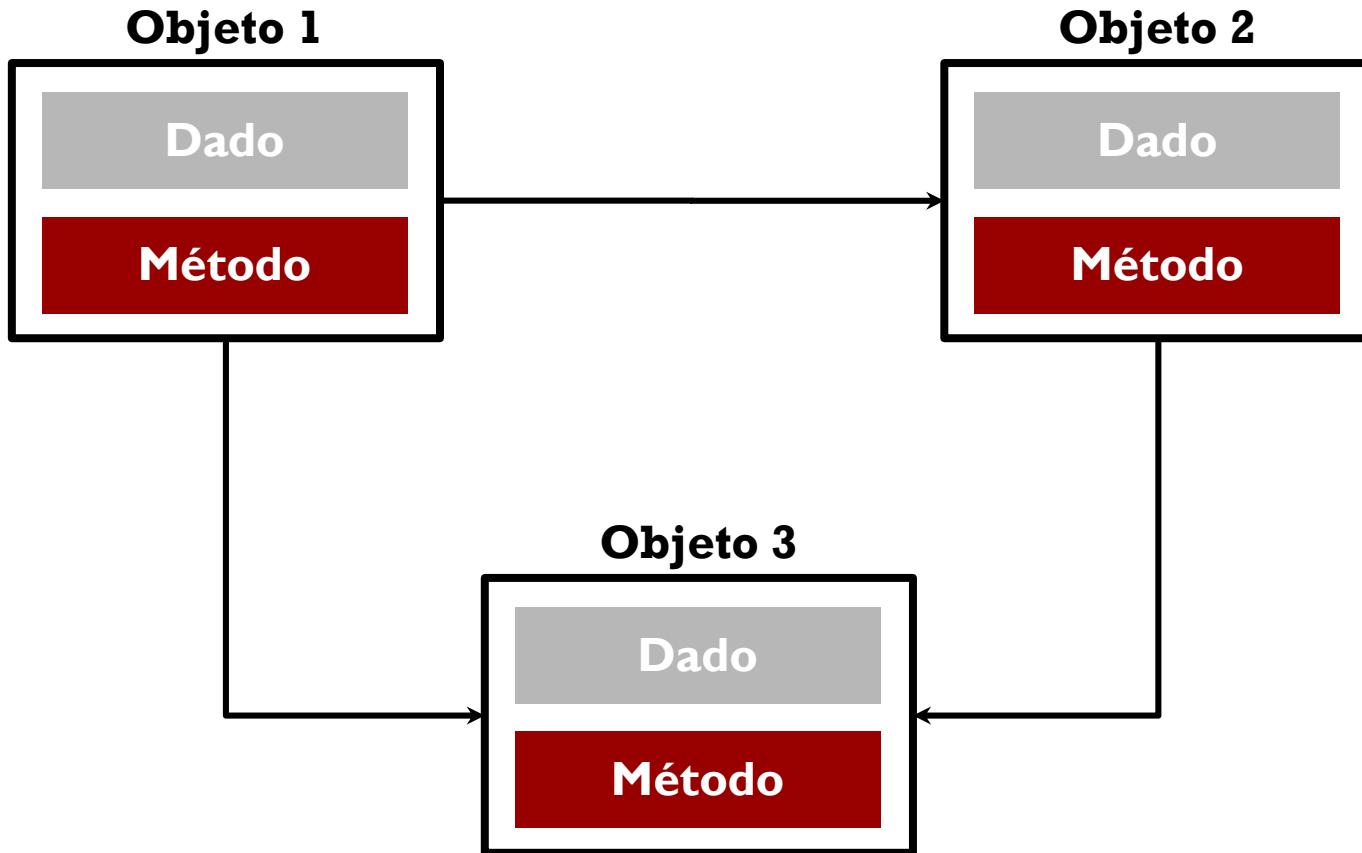


Programação Estruturada



Programação Orientada a Objetos

# Objetos se comunicam por mensagens



# Garantido a restrição de acesso

```
class Ponto {  
    private: Só pode ser acessado dentro da classe!  
        float _x;  
        float _y;  
    public: Acesso de fora da classe!  
        Ponto(float x, float y) {  
            _x = x;  
            _y = y;  
        }  
        float get_x() {  
            return _x;  
        }  
        float get_y() {  
            return _y;  
        }  
        void translacao(double dx, double dy) {  
            _x += dx;  
            _y += dy;  
        }  
};
```

# Limitando o acesso

## private vs public

- A palavra **private** garante que ninguém "de fora" bagunce seu objeto
- Para fazer uso do mesmo, tem que chamar os métodos **public**
- Qual a vantagem?

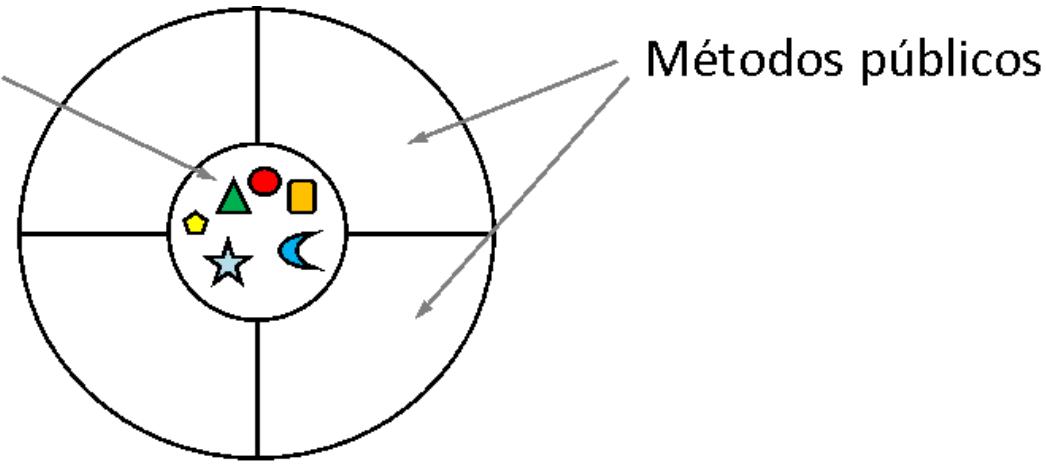
# Limitando o acesso

## private vs public

- A palavra **private** garante que ninguém "de fora" bagunce seu objeto
- Para fazer uso do mesmo, tem que chamar os métodos **public**
- Qual a vantagem?
  - Resto do código não pode bagunçar a memória
  - Software feito em pequenos pedaços

# Encapsulamento

Detalhes privativos  
de implementação



# Construindo objetos

```
class Ponto {  
private:  
    float _x;  
    float _y;  
public:  
    Ponto(float x, float y) {  
        _x = x;    ← Construtor: Define como os atributos serão acessados  
        _y = y;  
    }  
    float get_x() {  
        return _x;  
    }  
    float get_y() {  
        return _y;  
    }  
    void translacao(double dx, double dy) {  
        _x += dx;  
        _y += dy;  
    }  
};
```

# Construindo objetos

Também podemos usar o Heap

```
// . . . Declaração do ponto aqui em cima //
int main() {
    Ponto *ponto = new Ponto(7, 9);
    std::cout << "Valor de x: " << ponto->get_x() << std::endl;
    std::cout << "Valor de y: " << ponto->get_y() << std::endl;
    ponto->translacao(3, 1);
    std::cout << "Valor de x: " << ponto->get_x() << std::endl;
    std::cout << "Valor de y: " << ponto->get_y() << std::endl;
    delete ponto;
}
```

# Construindo objetos

Acesso por ->

```
// . . . Declaração do ponto aqui em cima //
int main() {
    Ponto *ponto = new Ponto(7, 9);
    std::cout << "Valor de x: " << ponto->get_x() << std::endl;
    std::cout << "Valor de y: " << ponto->get_y() << std::endl;
    ponto->translacao(3, 1);
    std::cout << "Valor de x: " << ponto->get_x() << std::endl;
    std::cout << "Valor de y: " << ponto->get_y() << std::endl;
    delete ponto;
}
```

# **Conceitos que vamos aprender**

---

# Programação Orientada a Objetos

## Princípios

- Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo
  - Modularidade
  - Mensagens
- 
- Princípios fundamentais

# Programação Orientada a Objetos

## Princípios - Abstração

- Modelagem de um domínio
  - Identificar artefatos de software
  - Ignorar aspectos não relevantes
  - Representação de detalhes relevantes do domínio do problema na linguagem de solução
- Classes são abstrações de conceitos

# Programação Orientada a Objetos

## Princípios - Encapsulamento

- Agrupamento dos dados e procedimentos correlacionados em uma mesma entidade
- Um sistema orientado a objetos baseia-se no contrato, não na implementação interna
- Proteção da estrutura interna (integridade)

# Programação Orientada a Objetos

## Princípios - Herança

- Permite a hierarquização das classes
- Classe especializada (subclasse, filha)
  - Herda as propriedades (atributos e métodos)
  - Pode sobrescrever/estender comportamentos
- Auxilia no reuso de código

# Programação Orientada a Objetos

## Princípios - Polimorfismo

- Tratar tipos diferentes de forma homogênea
- Classes distintas com métodos homônimos
- Diferentes níveis na mesma hierarquia
- Um método assume “diferentes formas”
- Apresenta diferentes comportamentos

# Programação Orientada a Objetos

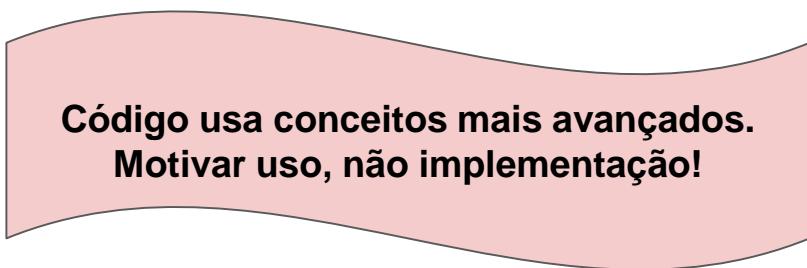
## Princípios - Mensagens

- **Comunicação entre objetos**
  - Envio/recebimento de mensagens
  - Forma de invocar um comportamento
- **Informação contida na mensagem**
  - Utiliza o contrato firmado entre as partes

# **Exemplo do Banco PDS2 (POO na Prática)**

<https://github.com/flaviovdf/programacao-2/tree/master/exemplos/aula04-encapsulamento/banco>

---



**Código usa conceitos mais avançados.  
Motivar uso, não implementação!**

# Exemplo de um main

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

#include "agencia.h"
#include "banco.h"
#include "cliente.h"
#include "conta.h"

int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667", "Pampulha",
                                            "Belo Horizonte", 3217901);
    // Adicionando um novo cliente
    agencia.adiciona_cliente(1, "Flavio Figueiredo");
    // Criando uma conta (no momento, 1 conta por cliente)
    Conta &conta = agencia.cria_conta(1);

    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
    conta.depositar(200);
    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
}
```

# Exemplo de um main

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

#include "agencia.h"
#include "banco.h"
#include "cliente.h"
#include "conta.h"

int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667", "Pampulha",
                                            "Belo Horizonte", 3217901);
    // Adicionando um novo cliente
    agencia.adiciona_cliente(1, "Flavio Figueiredo");
    // Criando uma conta (no momento, 1 conta por cliente)
    Conta &conta = agencia.cria_conta(1);

    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
    conta.depositar(200);
    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
}
```

Módulos que vou utilizar

Objetos

# Como fazer uso dos módulos?

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

#include "agencia.h"
#include "banco.h"
#include "cliente.h"
#include "conta.h"

int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667", "Pampulha",
                                            "Belo Horizonte", 3217901);
    // Adicionando um novo cliente
    agencia.adiciona_cliente(1, "Flavio Figueiredo");
    // Criando uma conta (no momento, 1 conta por cliente)
    Conta &conta = agencia.cria_conta(1);

    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
    conta.depositar(200);
    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
}
```

Módulos que vou utilizar

Objetos

# Chamada de função em POO

Cada objeto tem um estado, a função opera em cima do mesmo

```
int main(void) {  
    // Mesma coisa de Banco banco(1, "Banco do Brasil")  
    Banco banco = Banco(1, "Banco do Brasil");  
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667",  
                                            "Pampulha",  
                                            "Belo Horizonte", 3217901);  
}
```

# Chamada de função em POO

Neste momento entramos no construtor

```
int main(void) {  
    // Mesma coisa de Banco banco(1, "Banco do Brasil")  
    Banco banco = Banco(1, "Banco do Brasil");  
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667",  
                                            "Pampulha",  
                                            "Belo Horizonte", 3217901);  
}
```



# Chamada de função em POO

Neste momento entramos no construtor

```
int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667",
                                            "Pampulha",
                                            "Belo Horizonte", 3217901);
}
```

```
Banco::Banco(int numero, std::string nome) {
    _numero = numero;
    _nome = nome;
    _num_agencias = 0;
}
```



# Chamada de função em POO

Objeto é criado na memória

```
int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667",
                                            "Pampulha",
                                            "Belo Horizonte", 3217901);
}
```

```
Banco::Banco(int numero, std::string nome) {
    _numero = numero;
    _nome = nome;
    _num_agencias = 0;
}
```



# Memória

Um objeto é complicado

- Atributos + métodos
- Vamos representar de forma abstrar
  - Atributos apenas
  - Mora na pilha/stack do main neste caso

	<code>_numero</code>	<code>_nome</code>	<code>_num_agencias</code>
<code>main::banco (stack)</code>	1	"Banco do Brasil"	0

# Chamada de função em POO

## Chamando uma função de um objeto criado

```
int main(void) {  
    // Mesma coisa de Banco banco(1, "Banco do Brasil")  
    Banco banco = Banco(1, "Banco do Brasil");  
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667",  
                                            "Pampulha",  
                                            "Belo Horizonte", 3217901);  
}
```



# Chamada de função em POO

Entramos no método do objeto

```
int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667",
                                            "Pampulha",
                                            "Belo Horizonte", 3217901);
}
```

```
Agencia &Banco::cria_agencia(std::string logradouro, std::string bairro,
                               std::string cidade, int cep) {
    int numero = ++_num_agencias;
    // Resto do código omitido por clareza
}
```



# Chamada de função em POO

Faz uso do estado atual **deste** objeto

	<code>_numero</code>	<code>_nome</code>	<code>_num_agencias</code>
<code>main::banco (stack)</code>	1	"Banco do Brasil"	0

```
Agencia &Banco::cria_agencia(std::string logradouro, std::string bairro,  
                                std::string cidade, int cep) {  
    int numero = ++_num_agencias;  
    // Resto do código omitido por clareza  
}
```

# Chamada de função em POO

Faz uso do estado atual **deste** objeto

	<code>_numero</code>	<code>_nome</code>	<code>_num_agencias</code>
<code>main::banco (stack)</code>	1	"Banco do Brasil"	1

```
Agencia &Banco::cria_agencia(std::string logradouro, std::string bairro,  
                                std::string cidade, int cep) {  
    int numero = ++_num_agencias;  
    // Resto do código omitido por clareza  
}
```



# Múltiplos objetos

Nada impede de termos  $n > 1$  objetos com estados diferentes

```
int main(void) {
    Banco bb = Banco(1, "Banco do Brasil");
    Banco bradesco = Banco(2, "Bradesco");
}
```

	<u>numero</u>	<u>nome</u>	<u>num_agencias</u>
<b>main::bb (stack)</b>	1	"Banco do Brasil"	0
<b>main::bradesco (stack)</b>	2	"Bradesco"	0

# Múltiplos objetos

Nada impede de termos  $n > 1$  objetos com estados diferentes

```
int main(void) {
    Banco bb = Banco(1, "Banco do Brasil");
    Banco bradesco = Banco(2, "Bradesco");
    Agencia &agencia = bb.cria_agencia("Antonio Carlos, 6667",
                                         "Pampulha",
                                         "Belo Horizonte", 3217901);
}
```

	<u>_numero</u>	<u>_nome</u>	<u>_num_agencias</u>
<b>main::bb (stack)</b>	1	"Banco do Brasil"	<b>1</b>
<b>main::bradesco (stack)</b>	2	"Bradesco"	<b>0</b>

# Os objetos deste programa moram aonde?

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

#include "agencia.h"
#include "banco.h"
#include "cliente.h"
#include "conta.h"

int main(void) {
    // Mesma coisa de Banco banco(1, "Banco do Brasil")
    Banco banco = Banco(1, "Banco do Brasil");
    Agencia &agencia = banco.cria_agencia("Antonio Carlos, 6667", "Pampulha",
                                            "Belo Horizonte", 3217901);
    // Adicionando um novo cliente
    agencia.adiciona_cliente(1, "Flavio Figueiredo");
    // Criando uma conta (no momento, 1 conta por cliente)
    Conta &conta = agencia.cria_conta(1);

    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
    conta.depositar(200);
    std::cout << "Saldo de Flavio " << conta.get_saldo() << std::endl;
}
```

Módulos que vou utilizar

Objetos

# **Múltiplas Classes em um Projeto**

---

# Uma Classe

Modelando um vírus infectando pacientes

Um Vírus tem um:

**nome (string)**

Uma força de infecção:

**força (double)**

Infecta um paciente quando:

**força > resistência (do paciente)**

# Modularizando o código

## Passo I

- Criar o cabeçalho/header (.h)
  - O cabeçalho é o contrato da sua classe
    - Os usuários vão ler o mesmo
    - Não precisam entender como é implementado
- Aprender a separar:
  - Contratos de Comportamento

# Cabeçalho (virus.h)

```
#ifndef PDS2_VIRUS_H
#define PDS2_VIRUS_H

#include <string>

class Virus {
private:
    std::string _nome;
    double _forca;
public:
    Virus(std::string nome, double forca);
    std::string get_nome();
    double get_forca();
};

#endif
```

# Cabeçalho (virus.h)

```
#ifndef PDS2_VIRUS_H  
#define PDS2_VIRUS_H
```

Guarda de segurança. Evita módulos com o mesmo nome

```
#include <string>
```

```
class Virus {  
private:  
    std::string _nome;  
    double _forca;  
public:  
    Virus(std::string nome, double forca);  
    std::string get_nome();  
    double get_forca();  
};
```

Atributos Privado

Construtor

Métodos públicos

```
#endif
```

Fim da guarda!

# Note que não temos o corpo dos métodos

```
#ifndef PDS2_VIRUS_H  
#define PDS2_VIRUS_H  
  
#include <string>  
  
class Virus {  
private:  
    std::string _nome;  
    double _forca;  
public:  
    Virus(std::string nome, double forca);  
    std::string get_nome();  
    double get_forca();  
};  
  
#endif
```

Guarda de segurança. Evita módulos com o mesmo nome

Atributos Privado

Construtor

Métodos públicos

Fim da guarda!

# Classes

## Membros

- Tipos de componentes
  - Membros de instância
  - Membros de classe (estáticos)
    - Assunto futuro
  - Procedimentos de inicialização
  - Procedimentos de destruição

# Implementando o .cpp

## Passo 2

- [Geralmente] Cada .h tem um .cpp
  - Existem exceções
  - Exceção no exemplo do banco (endereco.h)
- No .cpp vai o código

# Arquivo .cpp. Implementa os métodos

```
#include "virus.h" ← Tem que incluir o .h

Virus::Virus(std::string nome, double forca) {
    _nome = nome;
    _forca = forca;
}

std::string Virus::get_nome() {
    return _nome;
}

double Virus::get_forca() {
    return _forca; ← Implementação do método
}
```

# Arquivo .cpp. Implementa os métodos

```
#include "virus.h"
```

Tem que incluir o .h

```
Virus::Virus(std::string nome, double forca) {  
    _nome = nome;  
    _forca = forca;  
}
```

```
std::string Virus::get_nome() {  
    return _nome;  
}
```

Indica de qual classe pertence o método.

```
double Virus::get_forca() {  
    return _forca;  
}
```

Implementação do método

# Boas práticas de .h

- É possível ter mais de uma classe por .h
- Por isso o uso de ::

# Boas práticas de .h

## □ Possível de fazer, porém **evitar**

```
#ifndef PDS2_DUAS_CLASSES_H
#define PDS2_DUAS_CLASSES_H

class Class1 {
private:
    int _atributo;
public:
    int get_atributo();
};

class Class2 {
private:
    int _atributo;
public:
    int get_atributo();
};

#endif
```

```
#include "duasclasses.h"

int Class1::get_atributo() {
    return _atributo;
};

int Class2::get_atributo() {
    return _atributo;
};
```

Mesmo nome porém de classes diferentes

# this

```
#ifndef PDS2_VIRUS_H
#define PDS2_VIRUS_H

#include <string>

class Virus {
private:
    std::string nome;
    double forca;
public:
    Virus(std::string nome,
          double forca);
    std::string get_nome();
    double get_forca();
};

#endif
```



```
#include "virus.h"

Virus::Virus(std::string nome, double forca) {
    this->nome = nome;
    this->forca = forca;
}

std::string Virus::get_nome() {
    return this->nome;
}

double Virus::get_forca() {
    return this->forca;
}
```

# this

"Função identidade" de um objeto

- Em linguagens OO é comum ter um atributo implícito
- Em C++ o nome do mesmo é **this**

# this

"Função identidade" de um objeto

- **this** é um ponteiro para o próprio objeto
- Útil quando os atributos têm o mesmo nome dos parâmetros do método

```
#include "virus.h"

Virus::Virus(std::string nome, double forca) {
    this->nome = nome;
    this->forca = forca;
}
```

# this

## Quando usar

- Tem pessoas que preferem sempre usar
- Fica a seu critério
  - O uso de `_nome` antes de atributos é apenas um atalho para evitar `this`.

```
#include "virus.h"

Virus::Virus(std::string nome, double forca) {
    this->nome = nome;
    this->forca = forca;
}
```

# this

Não seria melhor uma referência?

- Lembrando que ponteiros e referências são quase iguais
- Porém o ponteiro veio antes
  - Então this é um ponteiro. Uso de ->

```
#include "virus.h"

Virus::Virus(std::string nome, double forca) {
    this->nome = nome;
    this->forca = forca;
}
```

# Classe Paciente

Seguimos o mesmo exemplo da classe Virus

- Criar um .h
- Criar um .cpp

# paciente.h

```
#ifndef PDS2_PACIENTE_H
#define PDS2_PACIENTE_H

#include <string>

#include "virus.h"

class Paciente {
private:
    std::string _nome;
    double _resistencia;
    bool _infectado;
    Virus *_virus;
public:
    Paciente(std::string nome, double resistencia);
    Paciente(std::string nome, double resistencia, Virus *virus);
    bool esta_infectado();
    Virus *get_virus();
    std::string get_nome();
    void contato(Paciente &contato);
    void curar();
};

#endif
```

# Constructor

```
#ifndef PDS2_PACIENTE_H
#define PDS2_PACIENTE_H

// . . .

class Paciente {
private:
    // . . .
    Virus *_virus;    ← Lembrando que virus é um ponteiro
    // . . .
    // . . .
#endif
```

```
#include "paciente.h"

Paciente::Paciente(std::string nome, double resistencia) {
    _nome = nome;
    _resistencia = resistencia;
    _infectado = false;
    _virus = nullptr;   ← Iniciamos para nullptr
}
```

# Constructor

```
#ifndef PDS2_PACIENTE_H
#define PDS2_PACIENTE_H

// . . .

class Paciente {
private:
    // . . .
    bool _infectado;
    Virus *_virus; ← Duas variáveis mantém o estado. Podemos simplificar no futuro
    // . . .
    // . . .
#endif
```

```
#include "paciente.h"

Paciente::Paciente(std::string nome, double resistencia) {
    _nome = nome;
    _resistencia = resistencia;
    _infectado = false; ← Setamos para false
    _virus = nullptr;
}
```

# Segundo Construtor

- Podemos ter várias formas de construir o mesmo objeto
- Basta que tenha parâmetros diferentes
  - Overload de funções
  - Pode ser feito para qualquer método

```
#include "paciente.h"

Paciente::Paciente(std::string nome, double resistencia, Virus *virus) {
    _nome = nome;
    _resistencia = resistencia;
    _infectado = true;
    _virus = virus;    ← Paciente já infectado
}
```

# Lembrando

## Métodos

- Procedimentos que podem modificar ou apenas acessar os valores dos atributos
- Controle de visibilidade
  - Determinar membros disponíveis para acesso
- Sobrecarga (overloading)
  - Dois ou mais métodos com mesmo nome
  - Lista de parâmetros (tipos) deve ser diferente!

# Métodos

Focando no mais complicado (código no github)

- Temos um método que:
  - Recebe um outro objeto do mesmo tipo
  - Usa **this** para diferenciar o local da memória

```
void Paciente::contato(Paciente &contato) {  
    if (contato.esta_infectado() && !this->esta_infectado()) {  
        if (contato.get_virus()->get_forca() > _resistencia) {  
            _infectado = true;  
            _virus = contato.get_virus();  
        }  
    }  
}
```

# Métodos

Focando no mais complicado (código no github)

- Temos um método que:
  - Recebe um outro objeto do mesmo tipo
  - Usa **this** para diferenciar o local da memória

```
void Paciente::contato(Paciente &contato)
{
    if (contato.esta_infectado() && !this->esta_infectado()) {
        if (contato.get_virus()->get_forca() > _resistencia) {
            _infectado = true;
            _virus = contato.get_virus();
        }
    }
}
```

# Métodos

## Quando chamamos Paciente::contato

```
void main(void) {  
    → Virus *virus = new Virus("V1", 0.8);  
    Paciente p1("John", 0.2, virus);  
    Paciente p2("Paul", 0.3);  
    p2.contato(p1);  
    delete virus;  
}
```

Stack

Heap

# Métodos

## Quando chamamos Paciente::contato

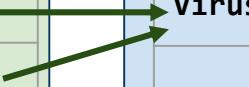
```
void main(void) {  
    Virus *virus = new Virus("V1", 0.8);  
    Paciente p1("John", 0.2, virus);  
    Paciente p2("Paul", 0.3);  
    p2.contato(p1);  
    delete virus;  
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016

Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}



# Métodos

## Quando chamamos Paciente::contato

```
void main(void) {  
    Virus *virus = new Virus("V1", 0.8);  
    Paciente p1("John", 0.2, virus);  
    Paciente p2("Paul", 0.3);  
    → p2.contato(p1);  
    delete virus;  
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016
main::p1	0x0036	{"John", 0.2, true, *_virus=0x0016}
main::p2	0x0028	{"Paul", 0.3, false, nullptr}

Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}

# Métodos

## Quando chamamos Paciente::contato

```
void Paciente::contato(Paciente &contato) {  
    if (contato.esta_infectado() && !this->esta_infectado()) {  
        if (contato.get_virus()->get_forca() > _resistencia) {  
            _infectado = true;  
            _virus = contato.get_virus();  
        }  
    }  
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016
main::p1	0x0036	{"John", 0.2, true, *_virus=0x0016}
main::p2	0x0028	{"Paul", 0.3, false, nullptr}

Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}

# Métodos

contato = p2; this = p1;

```
void Paciente::contato(Paciente &contato) {
    if (contato.esta_infectado() && !this->esta_infectado()) {
        if (contato.get_virus()->get_forca() > _resistencia) {
            _infectado = true;
            _virus = contato.get_virus();
        }
    }
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016
main::p1	0x0036	{"John", 0.2, true, *_virus=0x0016}
main::p2	0x0028	{"Paul", 0.3, false, nullptr}

Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}

# Métodos

contato = p2; this = p1;

```
void Paciente::contato(Paciente &contato) {  
    → if (contato.esta_infectado() && !this->esta_infectado()) {  
        if (contato.get_virus()->get_forca() > _resistencia) {  
            _infectado = true;  
            _virus = contato.get_virus();  
        }  
    }  
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016
main::p1	0x0036	{"John", 0.2, true, *_virus=0x0016}
main::p2	0x0028	{"Paul", 0.3, false, nullptr}

Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}

# Métodos

contato = p2; this = p1;

```
void Paciente::contato(Paciente &contato) {  
    if (contato.esta_infectado() && !this->esta_infectado()) {  
        → if (contato.get_virus()->get_forca() > _resistencia) {  
            _infectado = true;  
            _virus = contato.get_virus();  
        }  
    }  
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016
main::p1	0x0036	{"John", 0.2, true, *_virus=0x0016}
main::p2	0x0028	{"Paul", 0.3, false, nullptr}

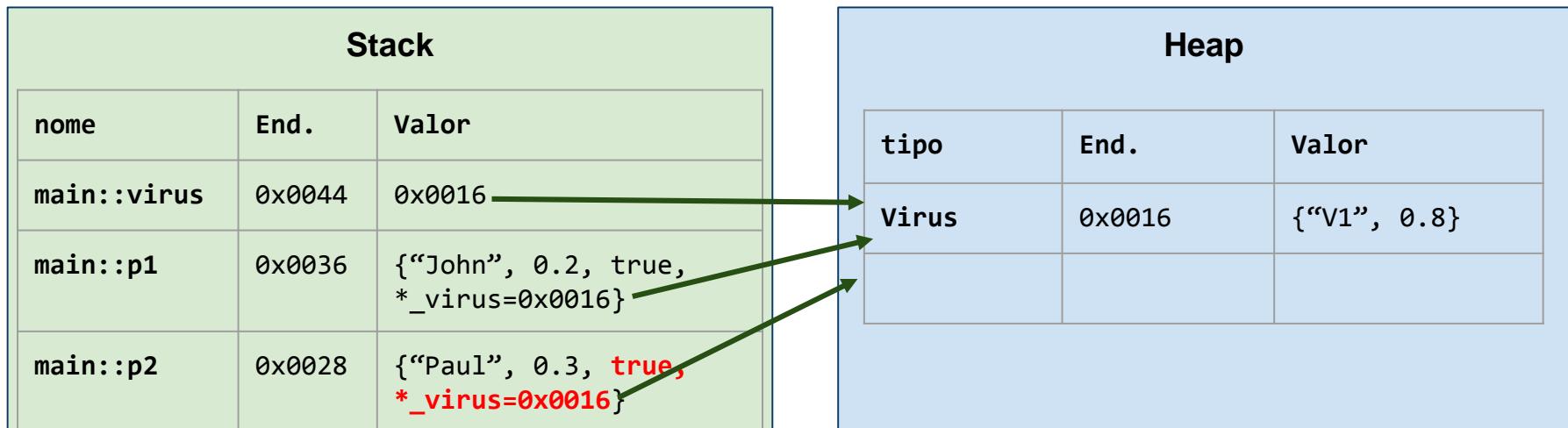
Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}

# Métodos

contato = p2; this = p1;

```
void Paciente::contato(Paciente &contato) {  
    if (contato.esta_infectado() && !this->esta_infectado()) {  
        if (contato.get_virus()->get_forca() > _resistencia) {  
            _infectado = true;  
            _virus = contato.get_virus();  
        }  
    }  
}
```



# Aquisição de Recurso é Inicialização

Se o main chamou **new** o main chama **delete**

```
void main(void) {  
    Virus *virus = new Virus("V1", 0.8);  
    Paciente p1("John", 0.2, virus);  
    Paciente p2("Paul", 0.3);  
    p2.contato(p1);  
    delete virus;  
}
```

Stack

nome	End.	Valor
main::virus	0x0044	0x0016
main::p1	0x0036	{"John", 0.2, true, *_virus=0x0016}
main::p2	0x0028	{"Paul", 0.3, <b>true</b> , <b>*_virus=0x0016</b> }

Heap

tipo	End.	Valor
Virus	0x0016	{"V1", 0.8}

# Programação e Desenvolvimento de Software 2

## Escopo, Cuidados da Memória e Static

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# Introdução

- Abstração
  - Simplificação de um problema difícil
  - É o ato de representar as características essenciais sem incluir os detalhes por trás
- Ocultação de dados
  - Informações desnecessárias devem ser escondidas do mundo externo (usuários)

# Lembrando

- Classes
  - Definem o comportamento dos objetos
  - Atributos, memória
  - Métodos, como os objetos são alterados
- Objetos
  - Instâncias na memória
  - Um estado
- Classes/Struct
  - Na classe tudo é **private** inicialmente

# **Escopo e Visibilidade**

---

# Ocultação I: Escopo

- O escopo de uma variável é a região de um programa dentro do qual a variável pode ser referenciada via de seu nome
- O escopo define quando o sistema aloca e libera memória para armazenar a variável
- Memória alocada no heap existem além do escopo da mesma

# Escopo

```
class MyClass {  
public:  
    int var1 ;           ← var1 e var2 tem escopo na classe  
    std::string var2;  
  
    void method(int param) { ← param tem escopo no método  
        int x = 1;          ← x e y tem escopo no method  
        int y = 9;  
        ...  
        if (param % 2 == 0) { ← res tem escopo no if  
            int res = 12;  
            return param * res;  
        }  
    }  
}
```

# Quando o método termina?

```
class MyClass {  
public:
```

```
    int var1 ;           ← var1 e var2 tem escopo na classe  
    std::string var2;
```

```
    void method(int param) { ← param tem escopo no método  
        int x = 1;  
        int y = 9;   ← x e y tem escopo no method
```

```
        ..  
        if (param % 2 == 0) { ← res tem escopo no if  
            int res = 12;  
            return param * res;  
        }
```

```
}
```

# x, y, res, param! "somem" da pilha

```
class MyClass {  
public:  
  
    int var1 ;           ← var1 e var2 tem escopo na classe  
    std::string var2;  
  
    void method(int param) { ← param tem escopo no método  
        int x = 1;          ← x e y tem escopo no method  
        int y = 9;  
        ...  
        if (param %2 == 0) { ← res tem escopo no if  
            int res = 12;  
            return param * res;  
        }  
    }  
}
```

# Quando o objeto é destruído?

```
class MyClass {  
public:
```

```
    int var1 ;           ← var1 e var2 tem escopo na classe  
    std::string var2;
```

```
    void method(int param) { ← param tem escopo no método  
        int x = 1;  
        int y = 9;      ← x e y tem escopo no method
```

```
        ..  
        if (param % 2 == 0) { ← res tem escopo no if  
            int res = 12;  
            return param * res;  
        }
```

```
}
```

# var1 e var2 somem

```
class MyClass {  
public:
```

```
    int var1 ;           ← var1 e var2 tem escopo na classe  
    std::string var2;
```

```
    void method(int param) { ← param tem escopo no método  
        int x = 1;  
        int y = 9;   ← x e y tem escopo no method  
        ...
```

```
        if (param % 2 == 0) { ← res tem escopo no if  
            int res = 12;  
            return param * res;  
        }
```

```
}
```

# Escopo e Estado

- O escopo define o **estado**
  - Estado do método
    - Valores das variáveis do método
  - Estado do objeto
    - Valores dos atributos
- Programação OO
  - Métodos
    - +
  - Estado do objeto

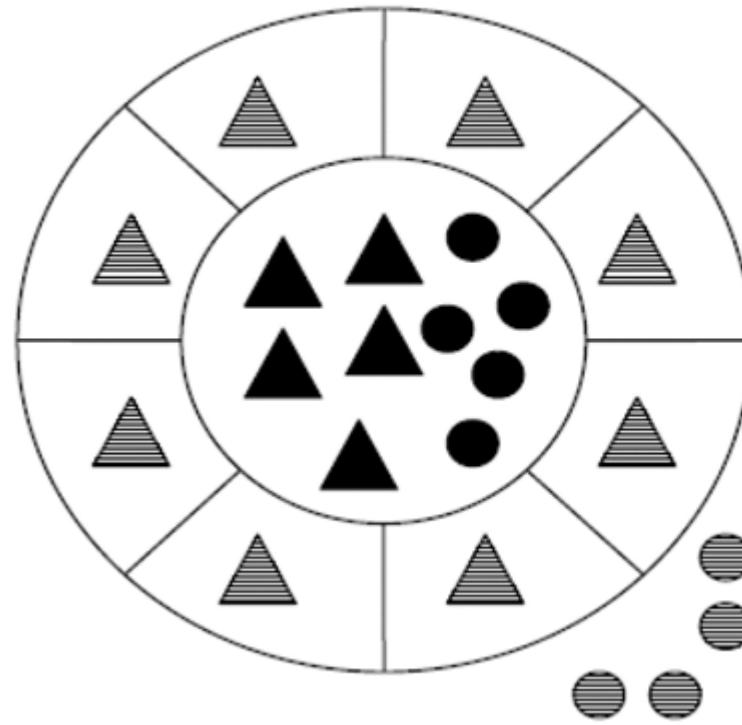
# Ocultação 2: Encapsulamento

- Encapsulamento
  - Mecanismo que coloca juntos os dados e suas funções associadas, mantendo-os controlados o acesso
- Proporciona abstração
  - Separa a visão externa da visão interna
  - Protege a integridade dos dados do Objeto

# Encapsulamento

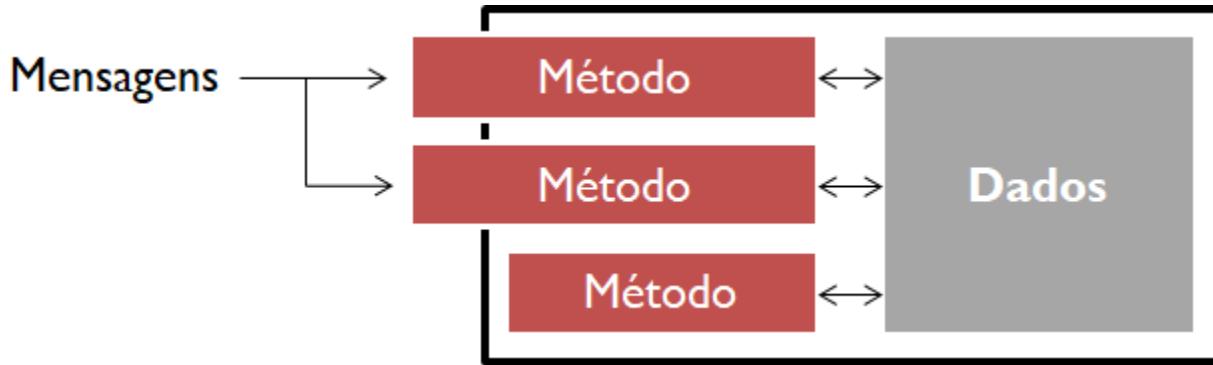
## CLASSE

- ▲ métodos públicos
- ▲ métodos privados
- dados privados
- dados públicos  
(não recomendável)



# Chamada de Métodos

- Informações encapsuladas em uma Classe
  - Estado (dados)
  - Comportamento (métodos)



# Encapsulamento

## Benefícios

### Desenvolvimento

- Melhor compreensão de cada classe
- Facilita a realização de testes

### Manutenção/Evolução

- Impacto reduzido ao modificar uma classe
- Interface deve ser o mais constante possível

# Encapsulamento

- Encapsulamento ocorre nas classes
- O comportamento e a interface de uma classe são definidos pelos seus membros
  - Atributos
  - Métodos
- Fazem uso dos modificadores de acesso

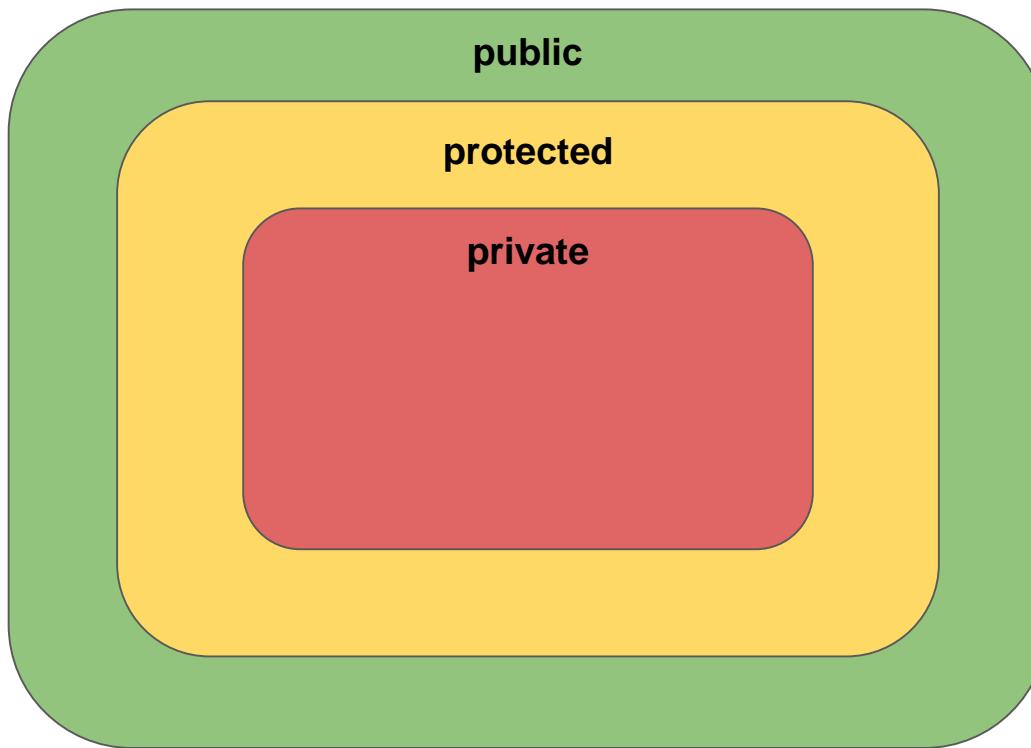
# Encapsulamento

C++

- Modificadores de acesso
  - Public
  - Protected
  - Private
- Membros declarados após o modificador

# Encapsulamento

## Modificadores de acesso



# Encapsulamento

## Modificadores de acesso - Public

- Permite que o membro público possa ser acessado de qualquer outra parte do código
- Mais liberal dos modificadores
  - Fazem parte (definem) o contrato da classe
  - Deve ser usado com responsabilidade
  - Não é recomendado
    - Por quê?

# Encapsulamento

## Modificadores de acesso - Public

- Qualquer outra classe acessa x, y
- Ferimos o encapsulamento
- Atributos public são raros, mas existem
  - Tenho certeza que mudar o mesmo não muda em nada a lógica de estado do objeto

```
class Ponto {  
public:  
    int x;  
    int y;  
};
```

# Encapsulamento

## Lei

- No sentido de não quebrar a encapsulação, é muito importante que os membros de uma classe (atributos e métodos) sejam visíveis apenas onde estritamente necessário
- A lei é: "**Não posso quebrar o que não posso acessar**"

# Encapsulamento

## Lei

- Por isso, é comum usarmos "**private**" como especificador de controle de acesso para atributos de uma classe
- Já que é freqüente querermos que métodos de uma classe sejam chamados por objetos de outras classes, não é raro usarmos "**public**" como especificador

# Encapsulamento

## Modificadores de acesso - Private

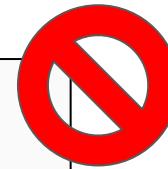
- Permite que o membro privado possa ser acessado por métodos da mesma classe
- O mais restritivo dos modificadores
  - Deve ser empregado sempre que possível
  - Utilizar métodos auxiliares de acesso
- Quando não há declaração explícita
  - Nível padrão (implícito)

# Encapsulamento

## Modificadores de acesso - Private

```
class Ponto {  
private:  
    int _x;  
    int _y;  
public:  
    Ponto(int x, int y):  
        _x(x), _y(y) {}  
};
```

```
int main() {  
    Ponto p(7, 10);  
    p._x = 9;  
}
```



# Encapsulamento

## Acessando e modificando atributos

- Evitar a manipulação direta de atributos
  - Acesso deve ser o mais restrito possível
  - De preferência todos devem ser private
- Sempre utilizar métodos auxiliares
  - Melhor controle das alterações
  - Acesso centralizado

# Encapsulamento

## Getters e Setters

- Convenção de nomenclatura dos métodos
- Get
  - Os métodos que permitem apenas o acesso de consulta (obter) devem possuir o prefixo get
- Set
  - Os métodos que permitem a alteração (definir) devem possuir o prefixo set

# Encapsulamento

## Getters e Setters

- Atributos **private** devem possuir get
  - set é opcional.
  - set CPF é raro.
- Nomenclatura alternativa
  - Atributos booleanos devem utilizar o prefixo “is” ao invés do prefixo get
- Melhora a legibilidade e entendimento

# Encapsulamento

## Modificadores de acesso - Protected

- Permite que o membro possa ser acessado apenas por outras classes que
  - Fazem parte da hierarquia (derivadas)
    - Próximas aulas
  - Classes “amigas”
    - Algo bem específico em C++

# **Destruitores**

---

# Ponteiros e Escopo

```
class MyClass {  
public:  
    int **var1 ;      quando que é liberado?!  
    std::string var2;
```

```
}
```

# Destruidores

Relembrando....

- Destruidores são chamados tanto para:
  - Objetos no heap
    - Depois de um delete
  - Objetos no stack
    - Depois que a função termina

**Cuide da memória que você alocou!**

# Problema

## Vírus que se reproduz

Vamos mudar o programa do Vírus para:

1. Garantir um limite de reprodução
  - Nunca se reproduz além do mesmo
2. Retornar uma cópia de si mesmo
  - Sempre que se reproduzir
3. Guardar ponteiros para as cópias
  - Assim rastrear todos os vírus
    - Alguns vírus têm mutações sazonais

# Exemplo

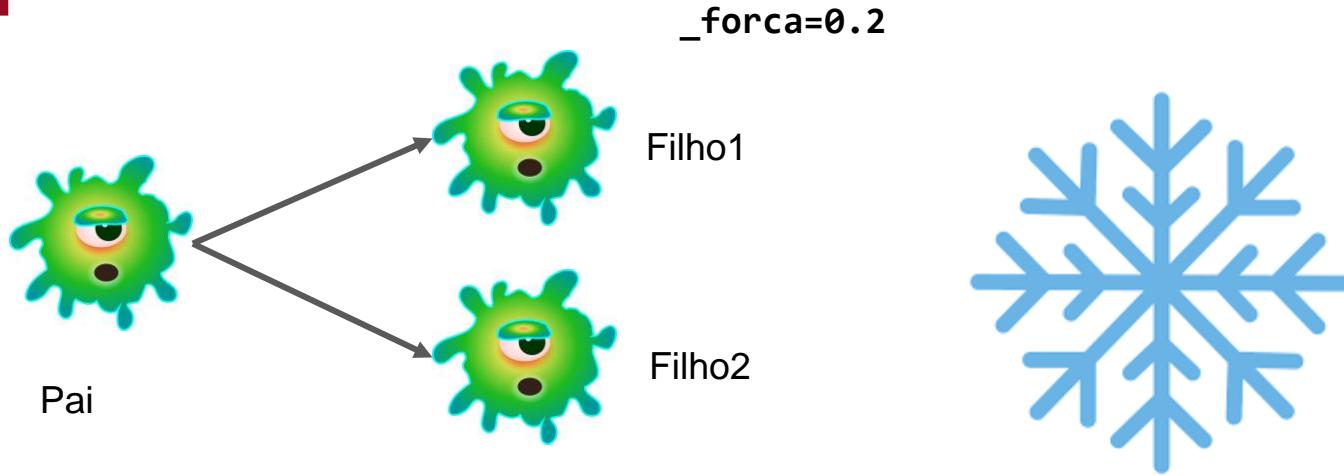
\_forca=0.2



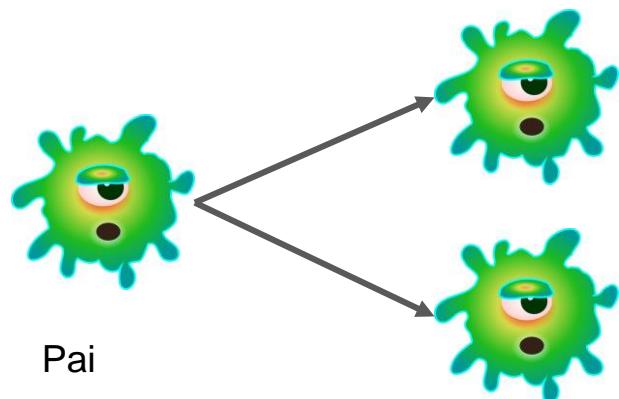
Pai



# Exemplo



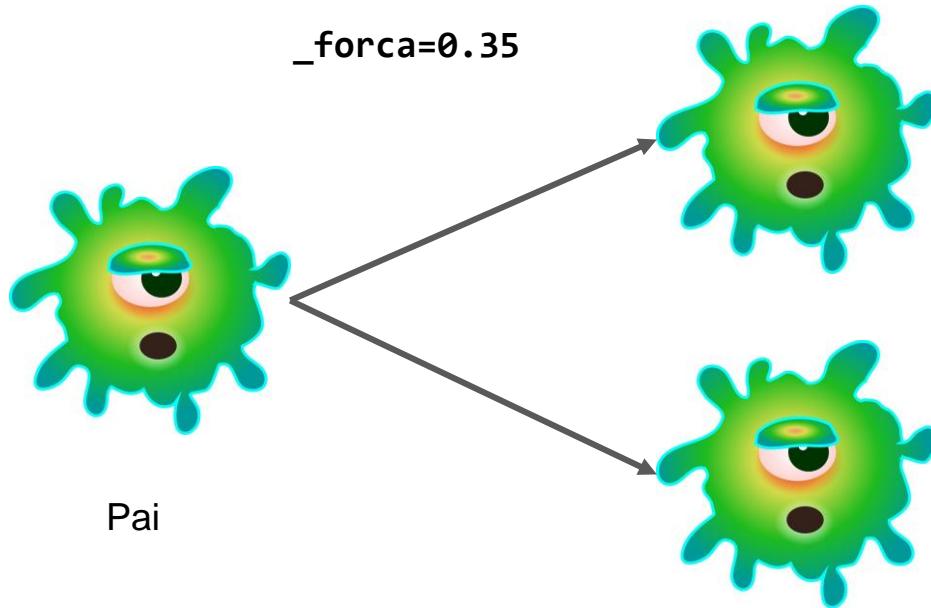
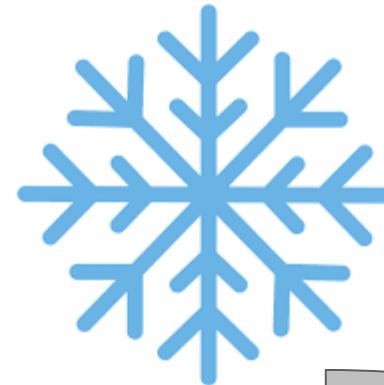
# Mudança de estação



\_forca=0.2

Filho1

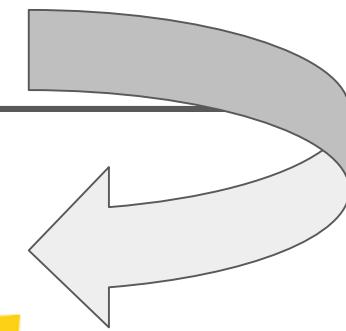
Filho2



\_forca=0.35

Filho1

Filho2



# virus.h

```
#ifndef PDS2_VIRUS_H
#define PDS2_VIRUS_H

#include <string>

class Virus {
private:
    // Guarda os filhos alocados. Todos no heap!
    Virus **_filhos;
    // Número de filhos atual
    int _numero_filhos;
    // . . . Mesmos atributos da aula anterior ... //
public:
    // Construtor. A capacidade de reprodução define o num filhos
    Virus(std::string nome, double forca, int capacidade_reproducao);
    // Destruitor para desalocar os filhos
    ~Virus();
    // . . . Mesmos métodos de antes ... //
    // Reproduz o vírus. Nossa foco!
    Virus *reproduzir();
};

#endif
```

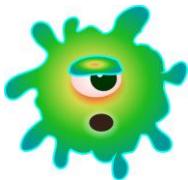
# Construtor

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    // vetor de tamanho capacidade_reproducao para os filhos (no heap)  
    _filhos = new Virus*[capacidade_reproducao]();  
}
```



# Construtor

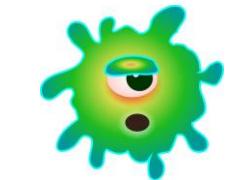
```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    → _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    // vetor de tamanho capacidade_reproducao para os filhos (no heap)  
    _filhos = new Virus*[capacidade_reproducao]();  
}
```



\_nome = "Bob"

# Construtor

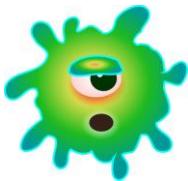
```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    // vetor de tamanho capacidade_reproducao para os filhos (no heap)  
    _filhos = new Virus*[capacidade_reproducao]();  
}
```



```
_nome = "Bob"  
_forca = 0.2
```

# Construtor

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    // vetor de tamanho capacidade_reproducao para os filhos (no heap)  
    _filhos = new Virus*[capacidade_reproducao]();  
}
```



```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4
```

# Construtor

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    // vetor de tamanho capacidade_reproducao para os filhos (no heap)  
    _filhos = new Virus*[capacidade_reproducao]();  
}
```



```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 0
```

# Constructor

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    // vetor de tamanho capacidade_reproducao para os filhos (no heap)  
    _filhos = new Virus*[capacidade_reproducao]();  
}
```

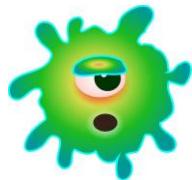


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 0  
\_filhos =



# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus;    // Guarda copia em um vetor
    _numero_filhos += 1;                      // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1];        // Retorna ponteiro para copia
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 0  
\_filhos =



# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus;    // Guarda copia em um vetor
    _numero_filhos += 1;                      // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1];        // Retorna ponteiro para copia
}
```

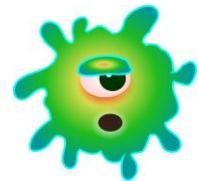


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 0  
\_filhos =



# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus;    // Guarda copia em um vetor
    _numero_filhos += 1;                      // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1];        // Retorna ponteiro para copia
}
```

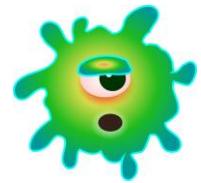


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 0  
\_filhos =



# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus;    // Guarda copia em um vetor
    _numero_filhos += 1;                      // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1];        // Retorna ponteiro para copia
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 0  
\_filhos =



# Reprodução

```
Virus *Virus::reproduzir() {  
    if (this->numero_filhos == this->capacidade_reproducao) {  
        return nullptr;  
    }  
    // Aloca uma um novo filho.  
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);  
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor  
    _numero_filhos += 1; // Aumenta o número de filhos  
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia  
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 1  
\_filhos =



# Reprodução

```
Virus *Virus::reproduzir() {  
    if (this->numero_filhos == this->capacidade_reproducao) {  
        return nullptr;  
    }  
    // Aloca uma um novo filho.  
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);  
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor  
    _numero_filhos += 1; // Aumenta o número de filhos  
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia  
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 1  
\_filhos =

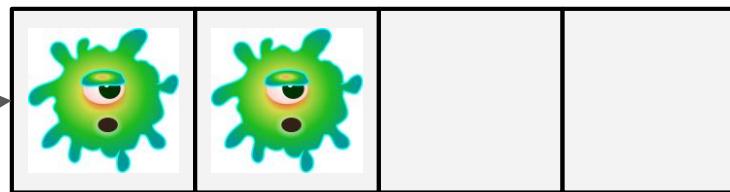


# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor
    _numero_filhos += 1; // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 1  
\_filhos =

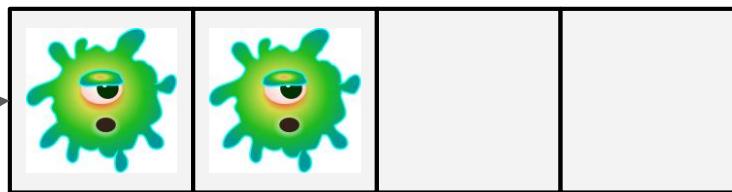


# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor
    _numero_filhos += 1; // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia
}
```

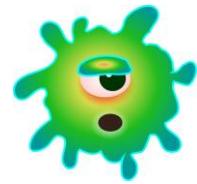


```
_nome = "Bob"
_forca = 0.2
_capacidade_reproducao = 4
_numero_filhos = 2
_filhos =
```

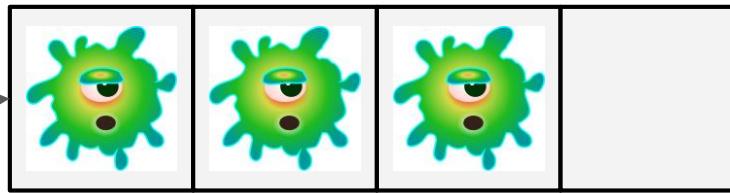


# Reprodução

```
Virus *Virus::reproduzir() {  
    if (this->numero_filhos == this->capacidade_reproducao) {  
        return nullptr;  
    }  
    // Aloca uma um novo filho.  
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);  
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor  
    _numero_filhos += 1; // Aumenta o número de filhos  
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia  
}
```

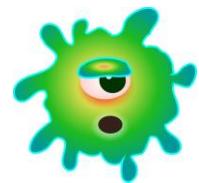


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 3  
\_filhos =

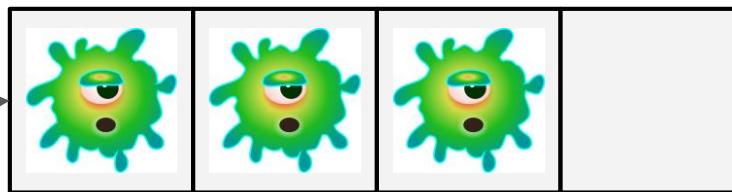


# Reprodução

```
Virus *Virus::reproduzir() {
    if (this->numero_filhos == this->capacidade_reproducao) {
        return nullptr;
    }
    // Aloca uma um novo filho.
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor
    _numero_filhos += 1; // Aumenta o número de filhos
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia
}
```

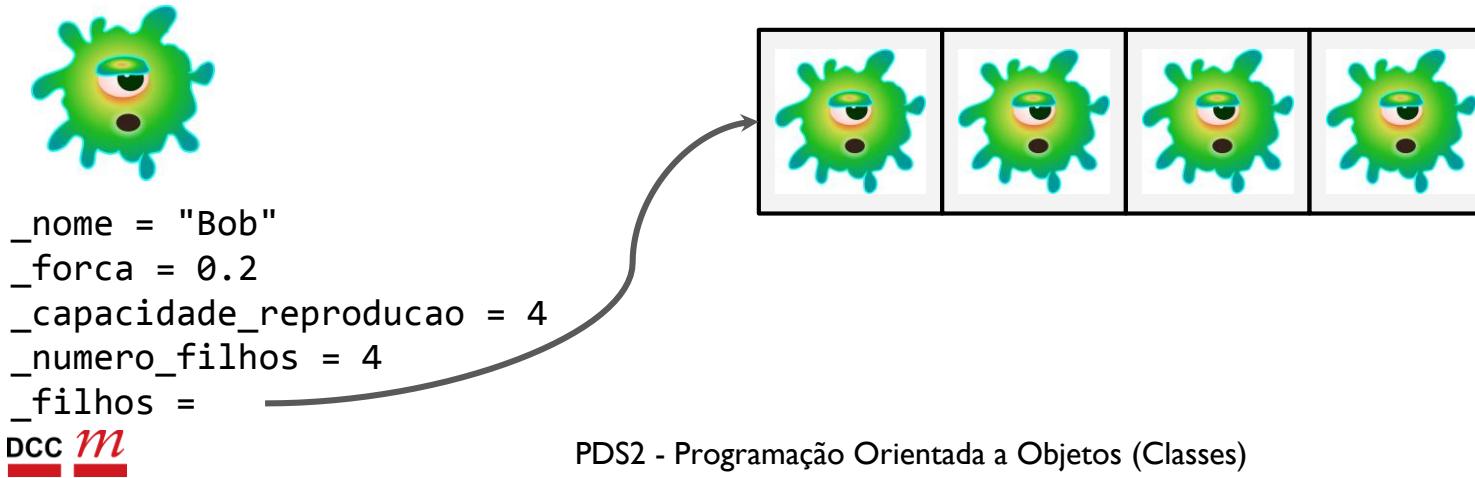


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 3  
\_filhos =



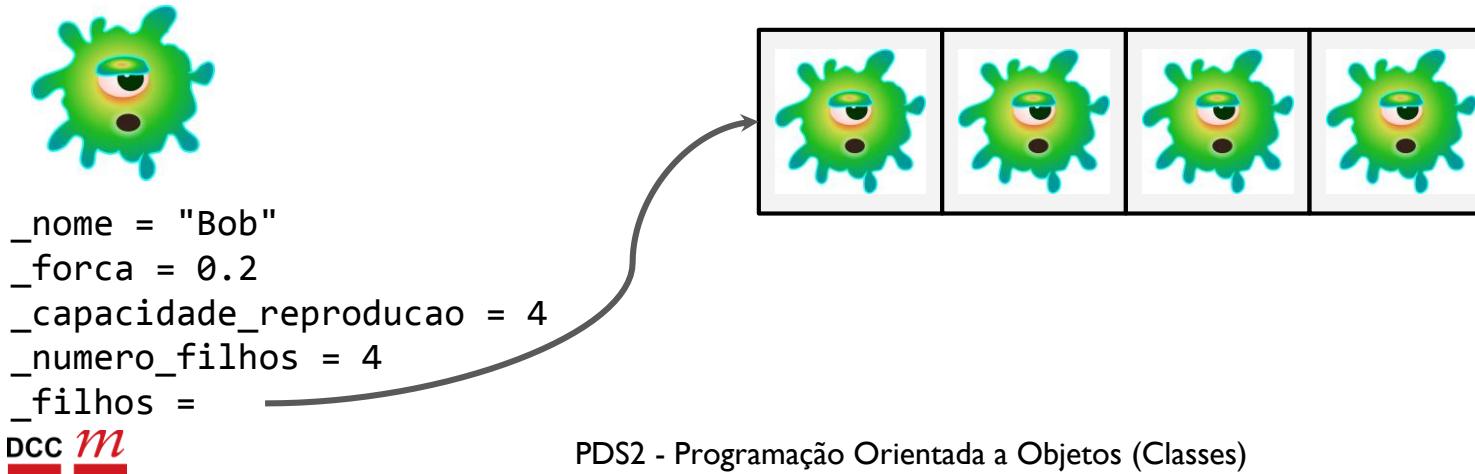
# Reprodução

```
Virus *Virus::reproduzir() {  
    if (this->numero_filhos == this->capacidade_reproducao) {  
        return nullptr;  
    }  
    // Aloca uma um novo filho.  
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);  
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor  
    _numero_filhos += 1; // Aumenta o número de filhos  
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia  
}
```



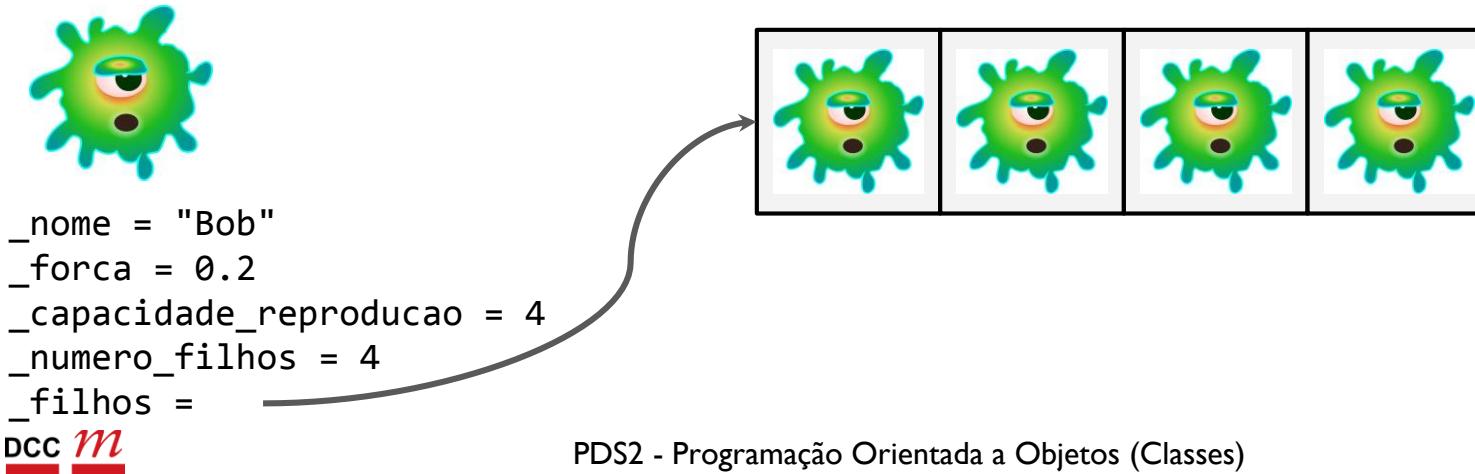
# Reprodução

```
Virus *Virus::reproduzir() {  
    if (this->numero_filhos == this->capacidade_reproducao) {  
        return nullptr;  
    }  
    // Aloca uma um novo filho.  
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);  
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor  
    _numero_filhos += 1; // Aumenta o número de filhos  
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia  
}
```



# Retornamos null indicando que não dá mais

```
Virus *Virus::reproduzir() {  
    if (this->numero_filhos == this->capacidade_reproducao) {  
        return nullptr;  
    }  
    // Aloca uma um novo filho.  
    Virus *novo_virus = new Virus(_nome, _forca, _capacidade_reproducao);  
    _filhos[_numero_filhos] = novo_virus; // Guarda copia em um vetor  
    _numero_filhos += 1; // Aumenta o número de filhos  
    return _filhos[_numero_filhos - 1]; // Retorna ponteiro para copia  
}
```

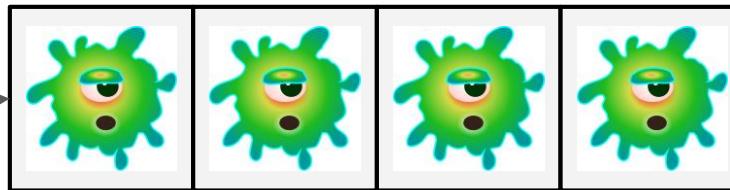


# Destruitor

```
Virus::~Virus() {  
    → if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```

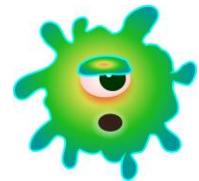


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

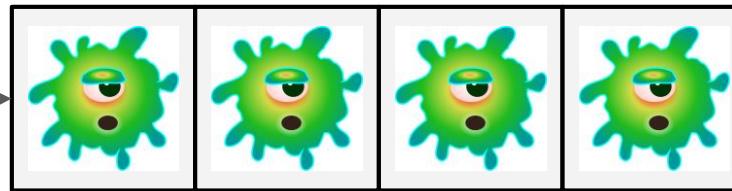


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        → for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
            delete[] _filhos;  
    }  
}
```

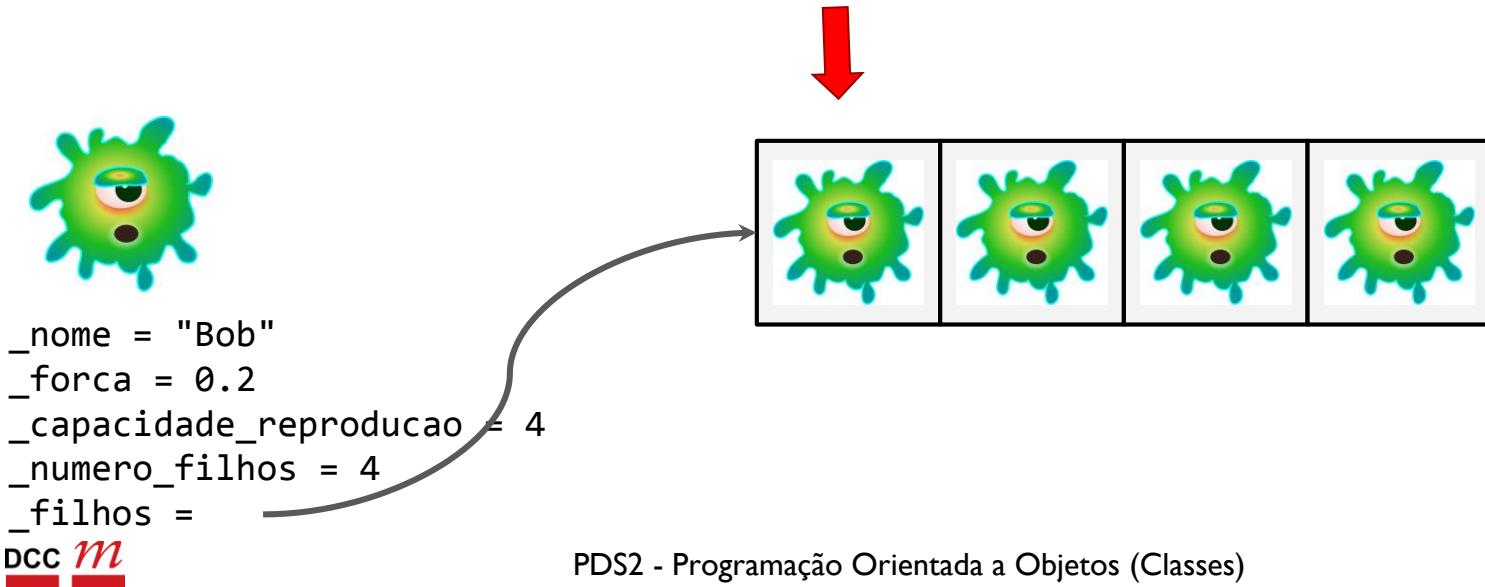


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =



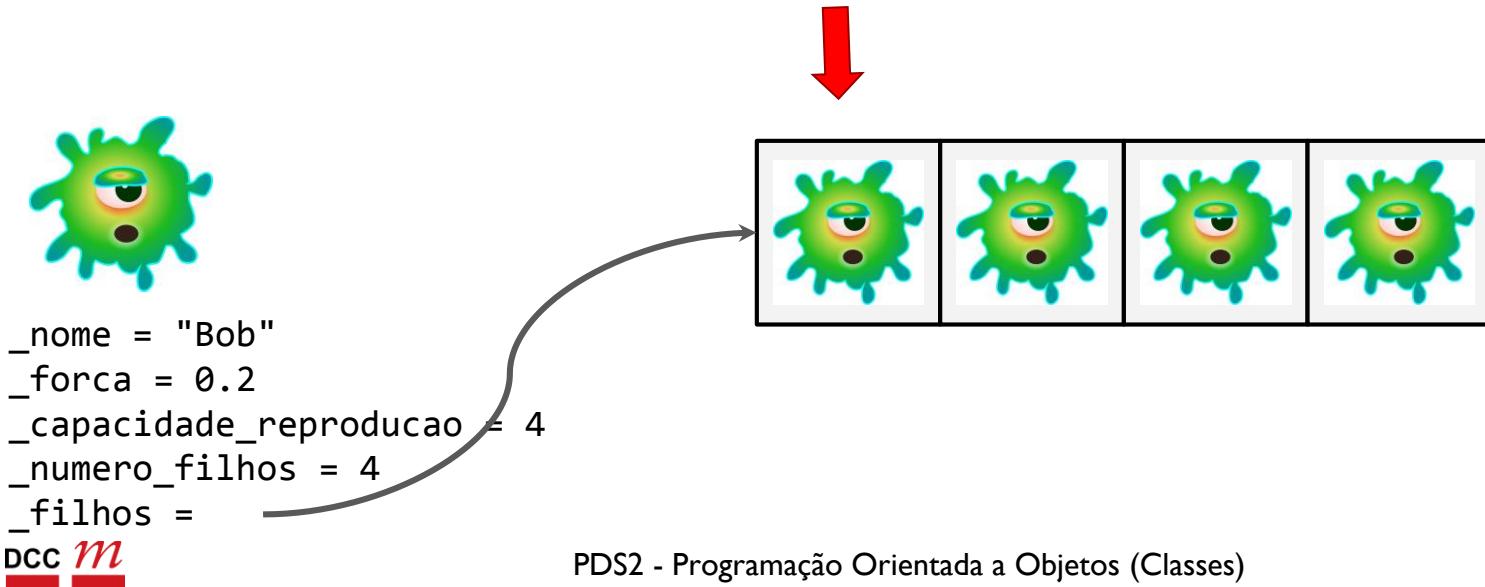
# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```

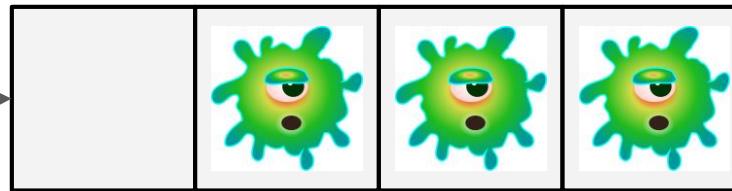


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

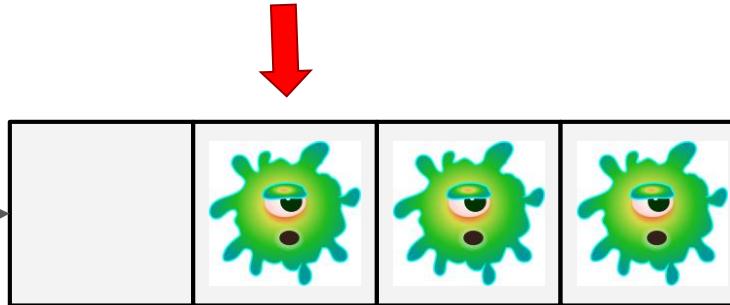


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

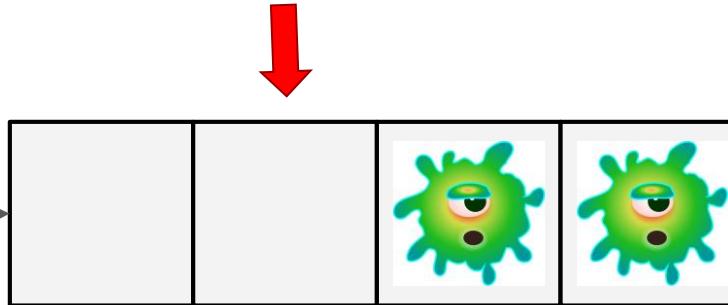


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```

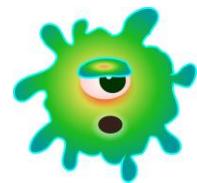


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

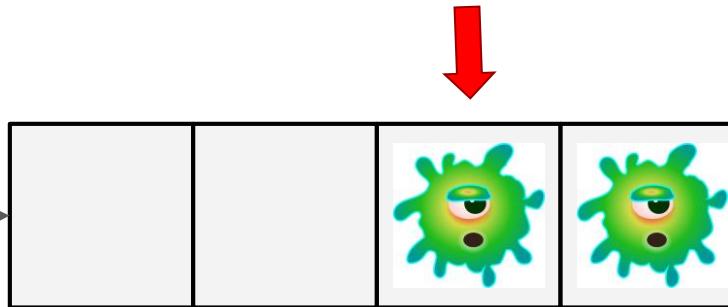


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```

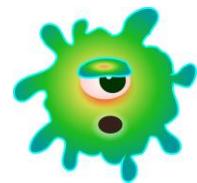


\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

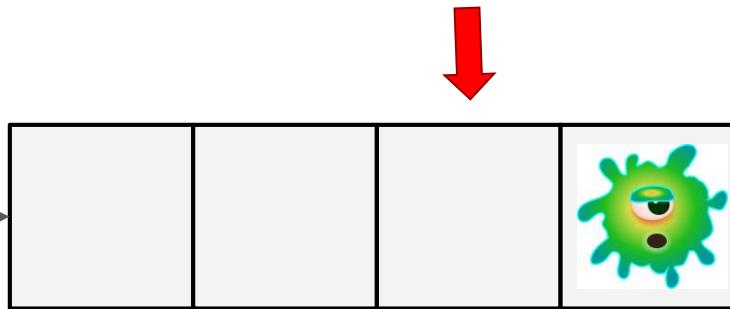


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

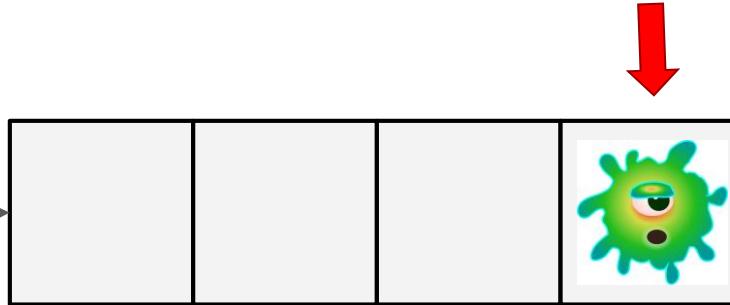


# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =

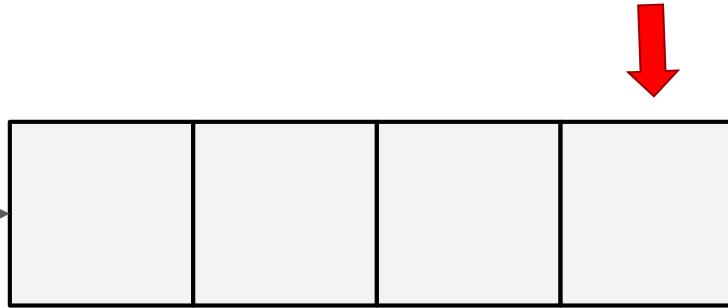


# Destruitor

```
Virus::~Virus() {
    if (_filhos != nullptr) {
        for (int i = 0; i < _numero_filhos; i++)
            if (_filhos[i] != nullptr)
                delete _filhos[i];
        delete[] _filhos;
    }
}
```



\_nome = "Bob"  
\_forca = 0.2  
\_capacidade\_reproducao = 4  
\_numero\_filhos = 4  
\_filhos =



# Destruitor

```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =`



# Destruitor

```
Virus::~Virus() {
    if (_filhos != nullptr) {
        for (int i = 0; i < _numero_filhos; i++)
            if (_filhos[i] != nullptr)
                delete _filhos[i];
        delete[] _filhos;
    }
}
```



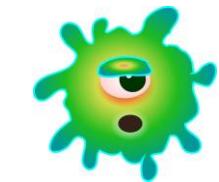
```
_nome = "Bob"
_forca = 0.2
_capacidade_reproducao = 4
_numero_filhos = 4
_filhos =
```

# Destruitor

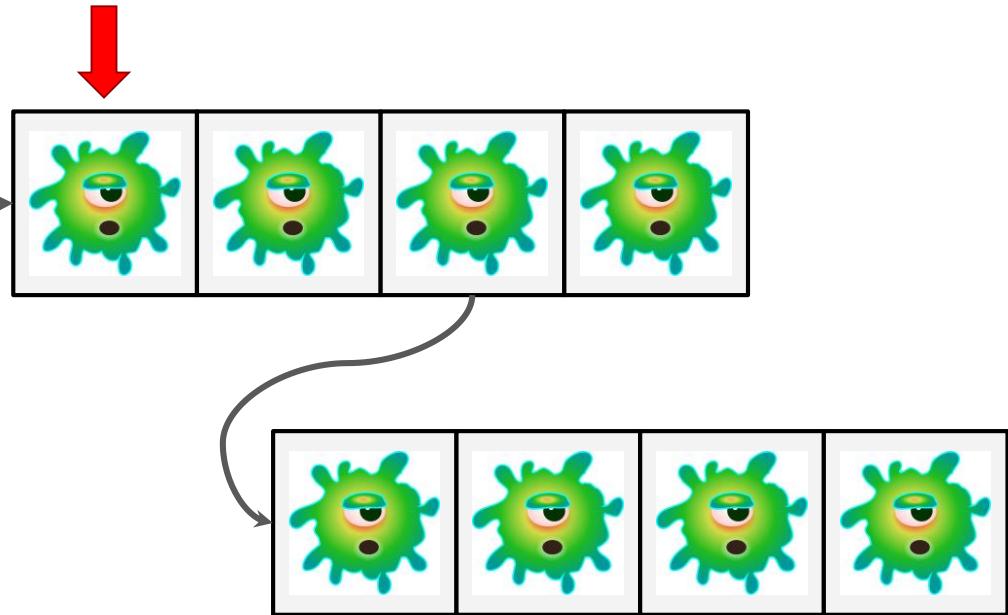
```
Virus::~Virus() {  
    if (_filhos != nullptr) {  
        for (int i = 0; i < _numero_filhos; i++)  
            if (_filhos[i] != nullptr)  
                delete _filhos[i];  
        delete[] _filhos;  
    }  
}
```



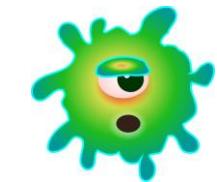
# Quando temos uma árvore



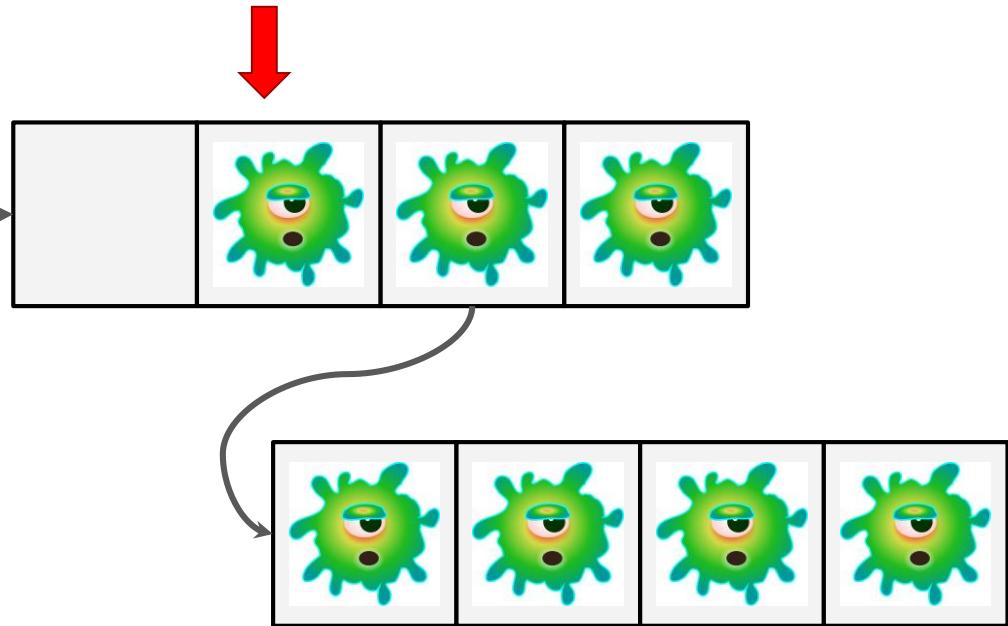
```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```



# Quando temos uma árvore



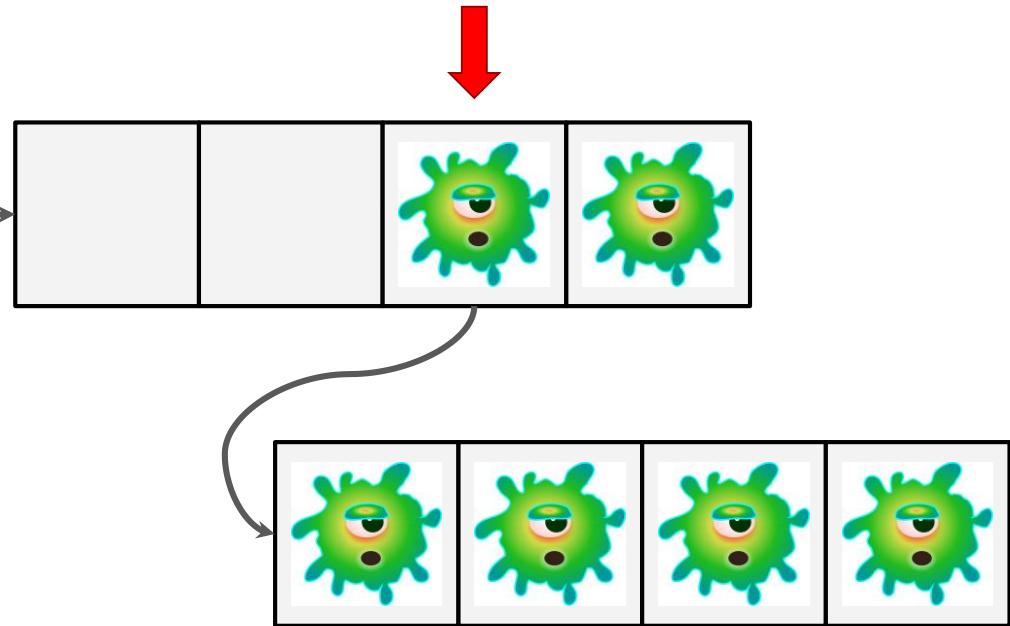
```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```



# Quando temos uma árvore

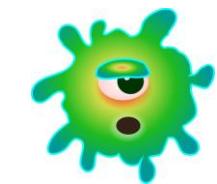


```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```

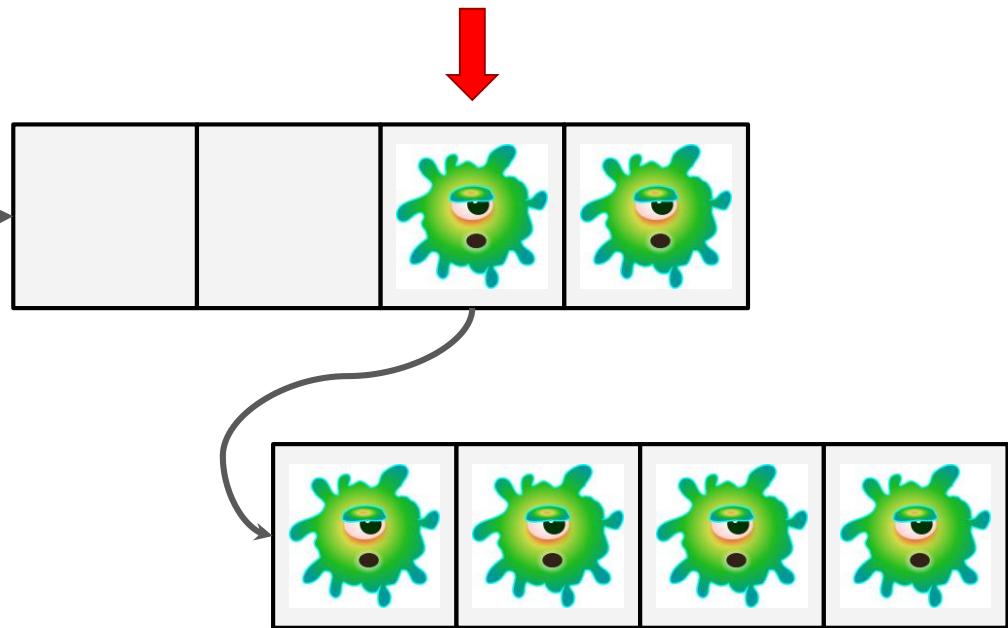


# O destrutor do vírus atual...

Também destrói sua lista de filhos

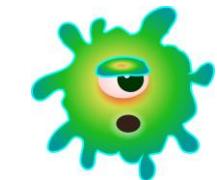


```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```

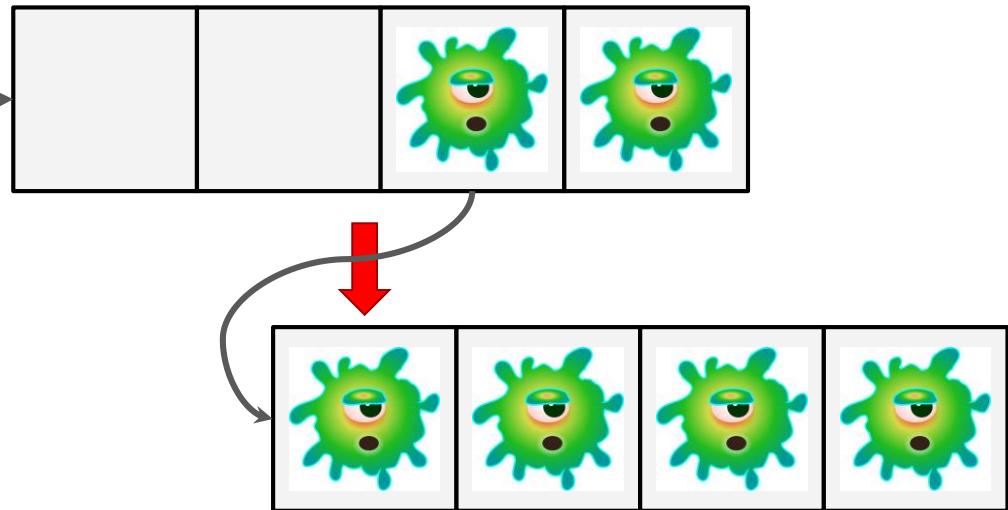


# O destrutor do vírus atual...

Também destrói sua lista de filhos

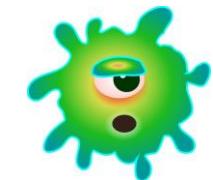


```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```

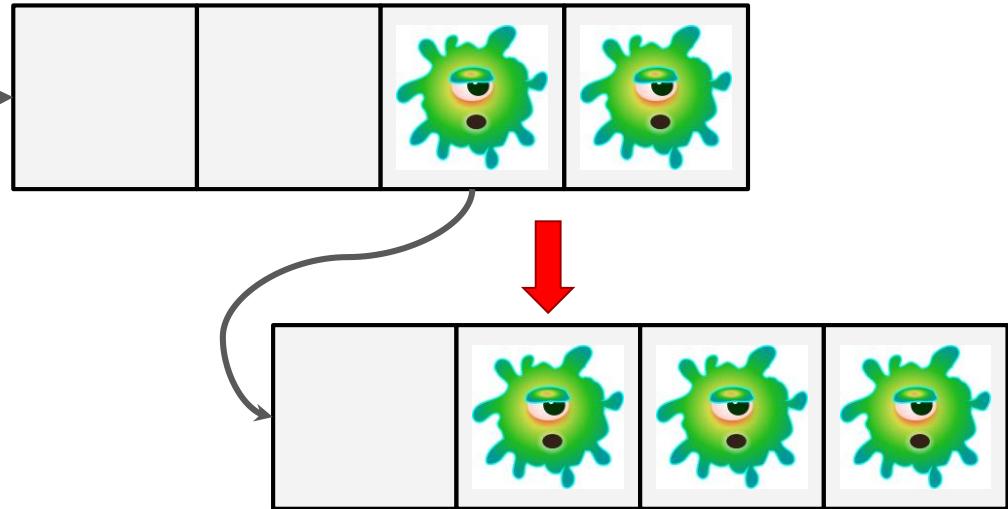


# O destrutor do vírus atual...

Também destrói sua lista de filhos

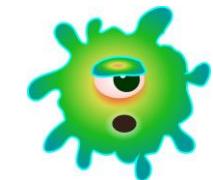


```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```

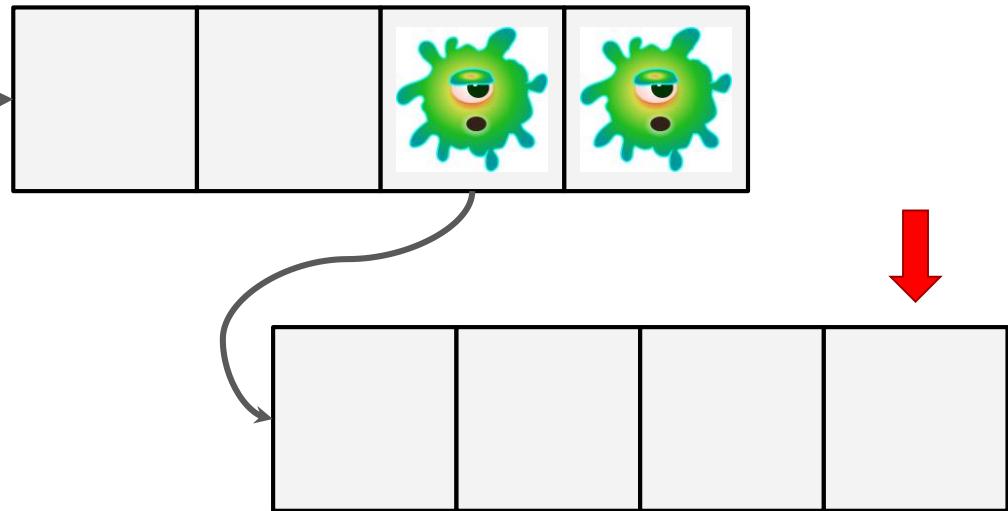


# O destrutor do vírus atual...

Também destrói sua lista de filhos



```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```

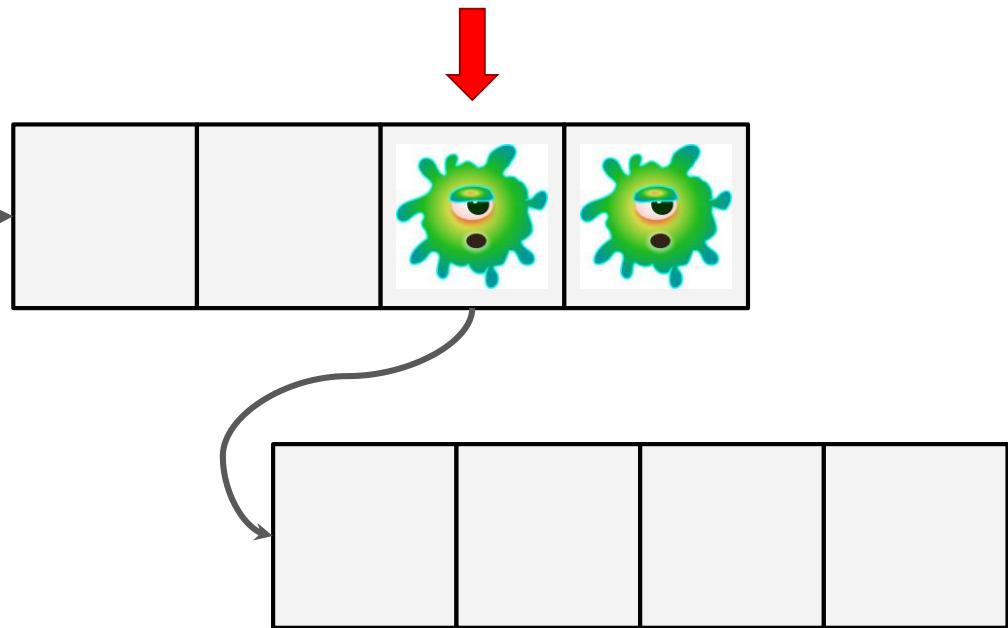


# O destrutor do vírus atual...

Também destrói sua lista de filhos

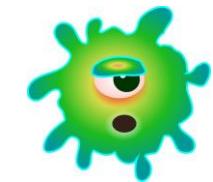


```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```

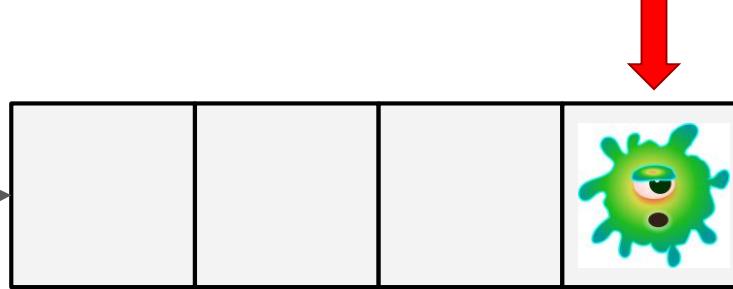


# O destrutor do vírus atual...

Também destrói sua lista de filhos



```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 4  
_filhos =
```



# Variáveis Estáticas

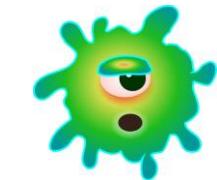
---

# Problema

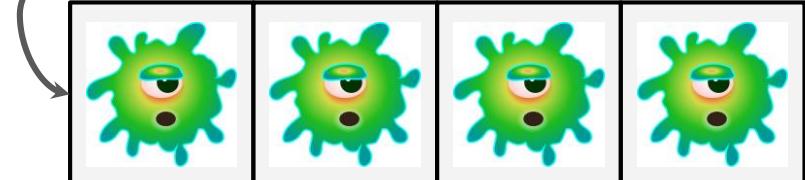
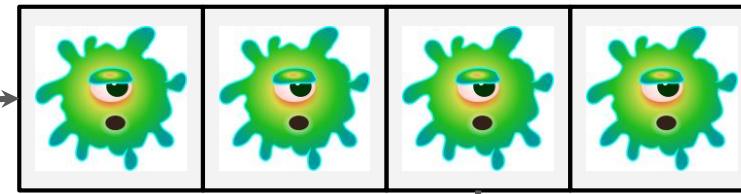
Contar a quantidade total de infecções de todos os vírus

Total é 9 no exemplo abaixo

Bob + 8 descendentes



```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos =  
_filhos =
```



# Classes

## Membros estáticos

- Membros de instância
  - Espaço de memória alocado para cada Objeto
  - Chamados somente através do Objeto
- **Membros de classe (estáticos)**
  - Espaço de memória único para todos Objetos
  - Podem ser chamados mesmo sem um Objeto

# Problema

## Alterando o .h

```
#ifndef PDS2_VIRUS_H
#define PDS2_VIRUS_H

class Virus {
private:
    // Conta quantas infecções todos os virus já causaram.
    static int _infeccoes_totais;
    // ...
public:
    // ...
    // Retorna quantas infecções todos os virus já causaram.
    static int get_infeccoes_totais();
};

#endif
```

# Problema

## Alterando o .h

```
#ifndef PDS2_VIRUS_H
#define PDS2_VIRUS_H

class Virus {
private:
    // Conta quantas infecções todos os virus já causaram.
    static int _infeccoes_totais;
    // ...
public:
    // ...
    // Retorna quantas infecções todos os virus já causaram.
    static int get_infeccoes_totais();
};

#endif
```

# Problema

## Alterando o .cpp

```
#include "virus.h"

int Virus::_infeccoes_totais = 0;

Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {
    _nome = nome;
    _forca = forca;
    _capacidade_reproducao = capacidade_reproducao;
    _numero_filhos = 0;
    _filhos = new Virus[capacidade_reproducao]();
    _infeccoes_totaais++;
}

// . . . todo o resto
```

# Problema

## Note as diferenças

```
#include "virus.h"

int Virus::_infeccoes_totais = 0; // Note que iniciamos fora do construto. Compartilhado!

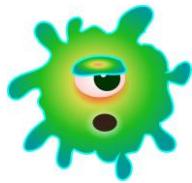
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {
    _nome = nome;
    _forca = forca;
    _capacidade_reproducao = capacidade_reproducao;
    _numero_filhos = 0;
    _filhos = new Virus[capacidade_reproducao]();
    _infeccoes_totais++; // Incrementamos ao criar novos virus
}

// . . . todo o resto
```

# Problema

## Passo a passo

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```

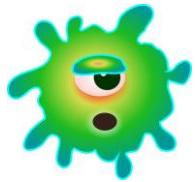


```
_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =  
Virus::_infeccoes_totaais = 1;
```

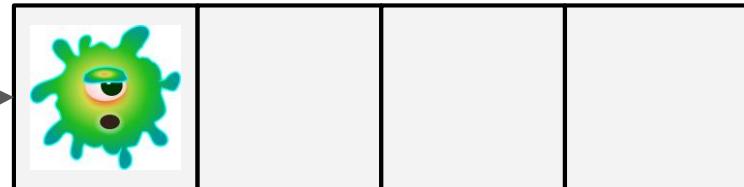
# Problema

Lembrando que o reproduzir chama o construtor

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```



`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =`

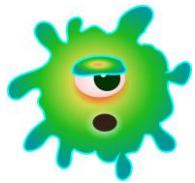


`Virus::_infeccoes_totaais = 2;`

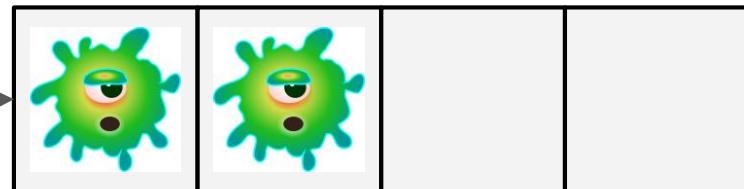
# Problema

Cada novo objeto incrementa o contador

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```



`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =`

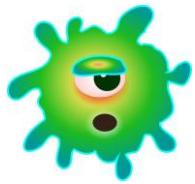


`Virus::_infeccoes_totaais = 3;`

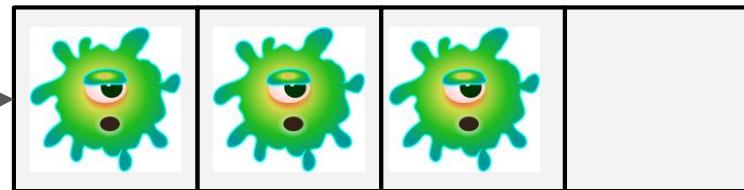
# Problema

Assim contamos o número total de vírus alocados

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```



`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =`



`Virus::_infeccoes_totaais = 4;`

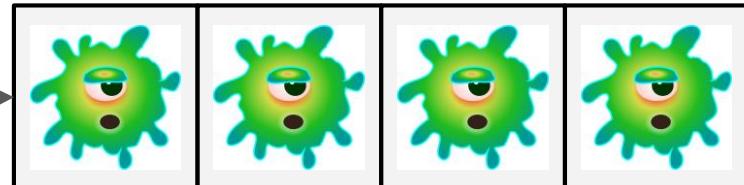
# Problema

Assim contamos o número total de vírus alocados

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```



`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =`



`Virus::_infeccoes_totaais = 5;`

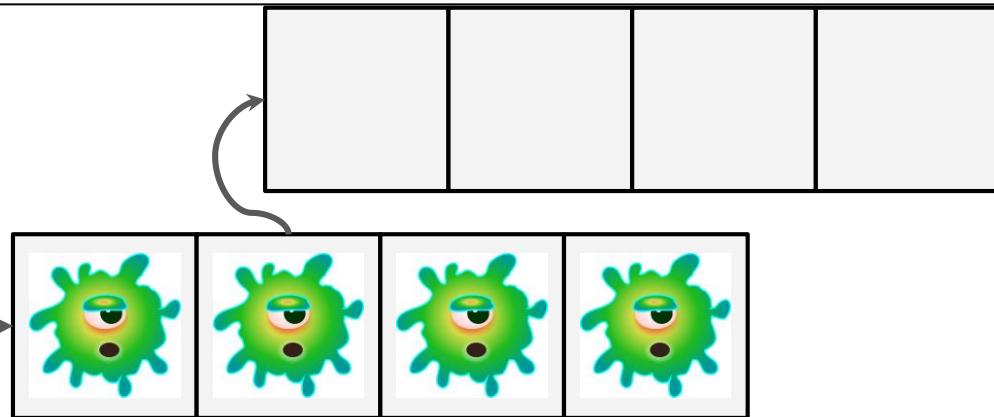
# Supondo que um dos filhos se reproduza

Como fica o contador?

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```



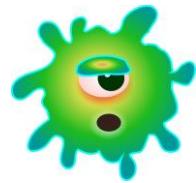
`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =`



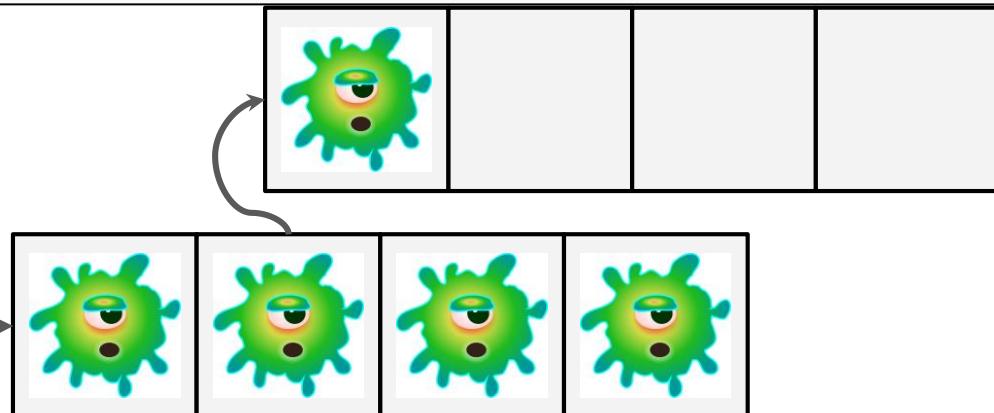
# Supondo que um dos filhos se reproduza

Continua ok! Todos os objetos da classe compartilham static

```
Virus::Virus(std::string nome, double forca, int capacidade_reproducao) {  
    _nome = nome;  
    _forca = forca;  
    _capacidade_reproducao = capacidade_reproducao;  
    _numero_filhos = 0;  
    _filhos = new Virus[capacidade_reproducao]();  
    _infeccoes_totaais++;  
}
```



`_nome = "Bob"  
_forca = 0.2  
_capacidade_reproducao = 4  
_numero_filhos = 1  
_filhos =`



`Virus::_infeccoes_totaais = 6;`

# Acessando o atributo estático

- O atributo pertence a todos os objetos
- Diferentes formas de acesso
- Todas são equivalentes
  - Acessa o mesmo local da memória

# Acessando o atributo estático

## □ Pela classe Virus::

```
int Virus::get_infeccoes_totais() {  
    return Virus::_infeccoes_totais;  
}
```

## □ Pelo objeto (com e sem this)

```
int Virus::get_infeccoes_totais() {  
    return _infeccoes_totais;  
}
```

```
int Virus::get_infeccoes_totais() {  
    return this->_infeccoes_totais;  
}
```

# Programação e Desenvolvimento de Software 2

## Conceito e especificação de software

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

# Introdução

- Introdução à Programação OO
  - O que é um objeto?
  - Como criar tais objetos?
- Requisito, análise e design
  - Quais objetos devem ser criados?
  - Como fazer os mesmos se comunicarem?
  - Quais atributos devem possuir

# Introdução

- Análise/Modelagem/Design
  - Quais objetos devem ser criados?
  - Quais características eles devem possuir?
  - Dados, comportamentos, ...

# Introdução

- Análise e Projeto
  - Quais as reais necessidades do cliente?
  - Requisitos → Software
- Software complexo → Planejamento
- Sistema mal projetado
  - Prejudica manutenção/extensão
  - Tempo → Dinheiro

# **Regras Iniciais**

---

# Aula de Hoje



Como o cliente explicou...



Como o líder de projeto entendeu...



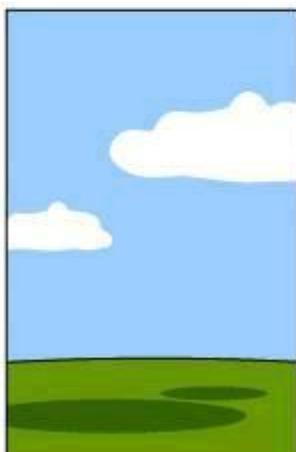
Como o analista projetou...



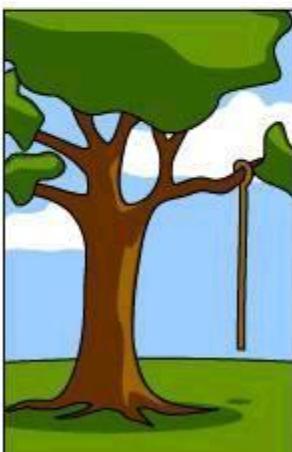
Como o programador construiu...



Como o Consultor de Negócios descreveu...



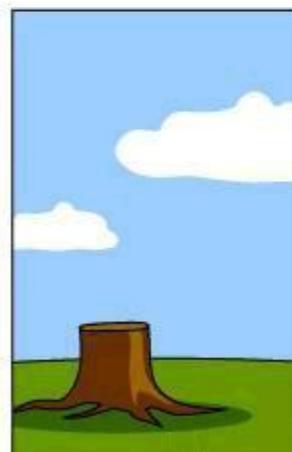
Como o projeto foi documentado...



Que funcionalidades foram instaladas...



Como o cliente foi cobrado...



Como foi mantido...



O que o cliente realmente queria...

# Duas Regras Simples

- Keep it Simple Stupid (KISS)!
  - Atingimos a perfeição não quando nada pode acrescentar-se a um projeto mas quando nada pode retirar-se.

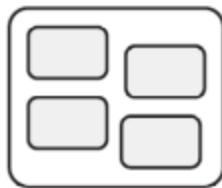
# Duas Regras Simples

- Keep it Simple Stupid (KISS)!
  - Atingimos a perfeição não quando nada pode acrescentar-se a um projeto mas quando nada pode retirar-se.
- Responsabilidade junto com os dados
  - Atribuir uma responsabilidade ao expert de informação - a classe que possui a informação necessária para preencher a responsabilidade

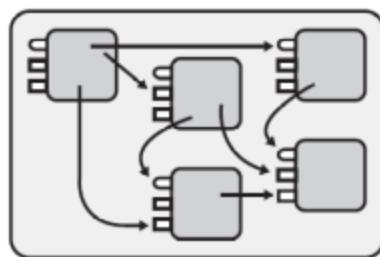
# Níveis de Modelagem



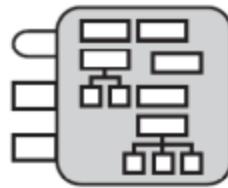
① Software system



② Division into subsystems/packages



③ Division into classes within packages



④ Division into data and routines within classes



⑤ Internal routine design

Entre as seguintes classes do mundo bancário, (Agencia, Conta, ContaCaixa, ContaSimples, Extrato, ExtratoHTML, Moeda, Movimento, Real, Transacao), quem deve ser responsável pela responsabilidade “Localizar a conta com certo número”?

# Especificação de Requisitos

- Requisitos
  - Objetivos e restrições estabelecidos por clientes e usuários que definem as propriedades desejadas
- Tipos
  - Funcionais -- Software tem que garantir
  - Não Funcionais -- Qualidades Globais

# User Stories

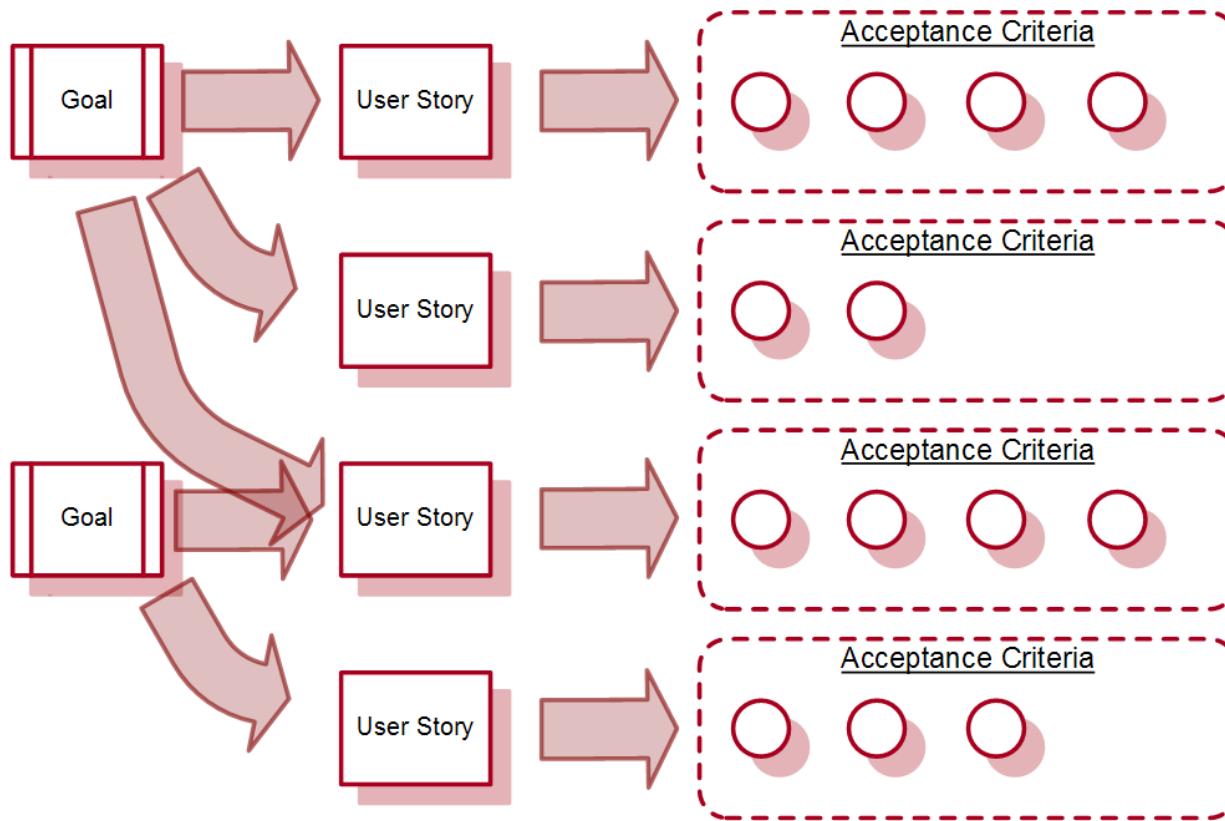
---

# User Stories

Kent Beck

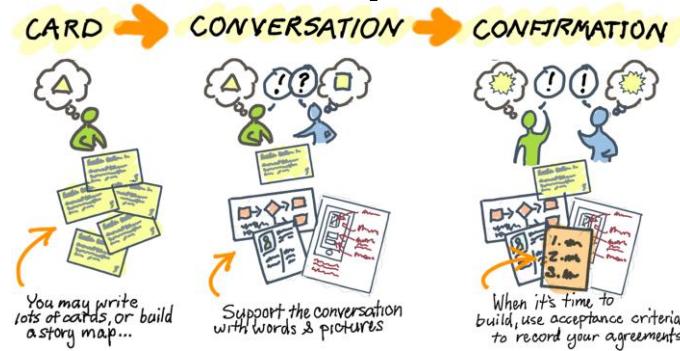
- Definição em linguagem natural de diferentes funcionalidades do programa
- Guiam o desenvolvimento do código
- Podemos desenvolver um user story por vez

# User Stories



# Cartões

- Cartão**
  - Descrição por escrito da história
- Conversão**
  - Discussão sobre a história
- Confirmação**
  - Descreva os testes que confirmam a história



# User Stories



**Como [Quem?] eu quero [O quê?] para [Por quê?].**

**Exemplo:** Como um cliente da operadora de saúde eu quero procurar um médico pelo nome para obter o endereço do seu consultório.

# Aceitação

- Critérios que precisam ser alcançados para que a User Story atenda os requisitos do usuário e seja aceita (concluída)
- Questões associadas a usabilidade, tratamento de erros, desempenho, funcionalidade, ...
- Devem apresentar a intenção, não a solução 3-5 testes/critérios por história

# Exemplo

Como um operador de marcação de consulta eu gostaria de visualizar a agenda de consultas dos médicos para saber quais horários estão disponíveis para novas marcações.

- Critérios de aceitação
  - Visualização da agenda por dia, semana e mês
  - Visualizar a agenda de consultas do mês atual e do mês posterior
  - Cada consulta agendada deve mostrar o nome e o convênio do paciente

# Exemplo

Como titular de um cartão de crédito, eu quero ver o extrato da minha conta para poder pagar o saldo devedor.

- Critérios de aceitação
  - Exibir saldo total
  - Exibir o pagamento mínimo devido
  - Exibir data de vencimento do pagamento
  - Exibir mensagem de erro se o serviço não estiver respondendo

# Exemplo

Como professor, eu gostaria de gerar um relatório de avaliação para poder avaliar o desempenho dos alunos.

- Critérios de aceitação
  - Mostrar a pontuação atual da avaliação de um aluno
  - Exibir a pontuação de avaliação passada do aluno
  - Fornecer uma opção para Imprimir/Salvar/Compartilhar

# Exercícios

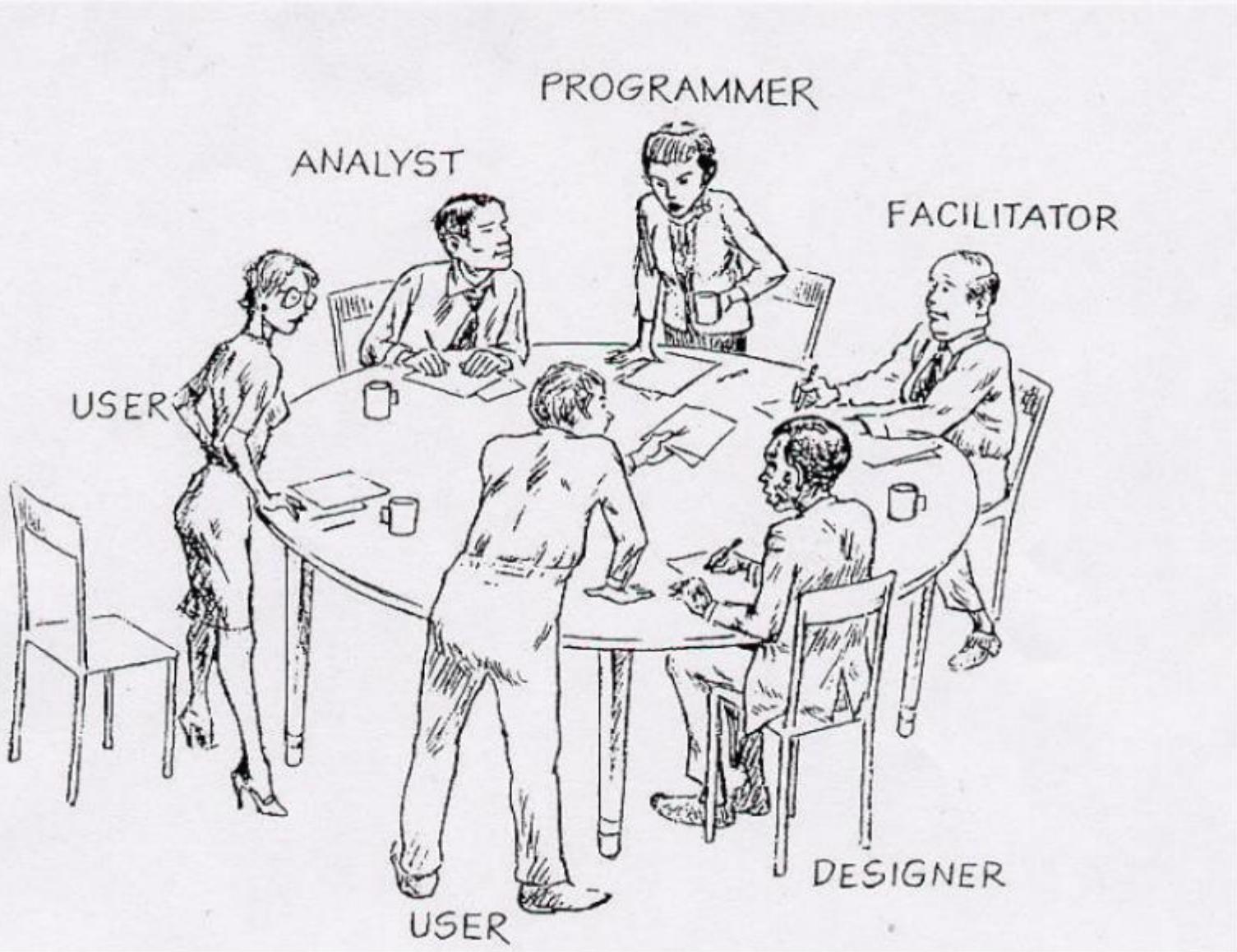
Definir User Stories para os seguintes casos

- Uber
  - Pedir um carro
- Netflix
  - Buscar um filme
- Waze
  - Procurar um caminho

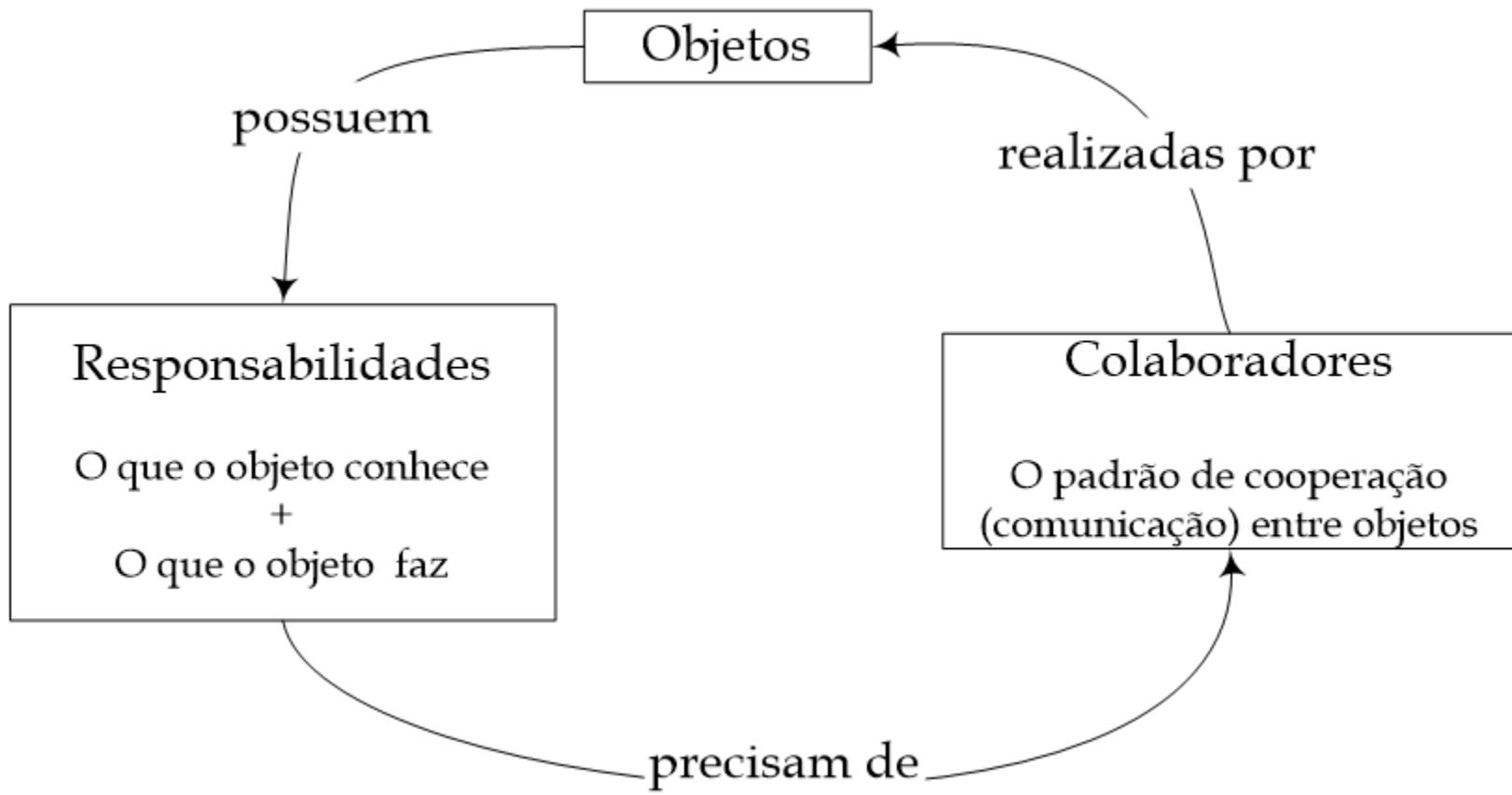
# **Cartões CRC**

---

# Sessões de Modelagem



# Modelagem do Mundo Através de Objetos



# Cartões CRC

- Kent Beck e Ward Cunningham (1989)
  - Introdução à *object-oriented thinking (design)*
- Ferramenta/Metodologia/Processo
  - Análise, modelagem e projeto de sistemas OO
- Nenhum processo garante bons resultados, mas é possível torná-los mais prováveis
  - Utilizar POO não é garantia de qualidade!

# Cartões CRC

- Cartões CRC
  - Class-Responsibility-Collaboration
  - 10cm x 15cm (físicos)
- Por que utilizar cartões?
  - Barato, portátil, disponível, familiar,  
...
  - Objetividade, simplicidade, clareza,  
...
  - Participação de diferentes atores

# Cartões CRC

Classe:

Responsabilidades

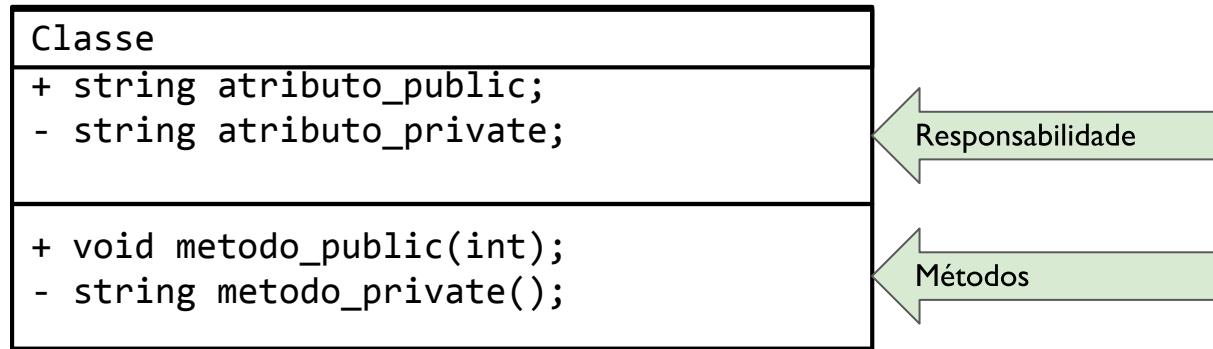
Colaborações

# **Modelos UML (simplificado)**

---

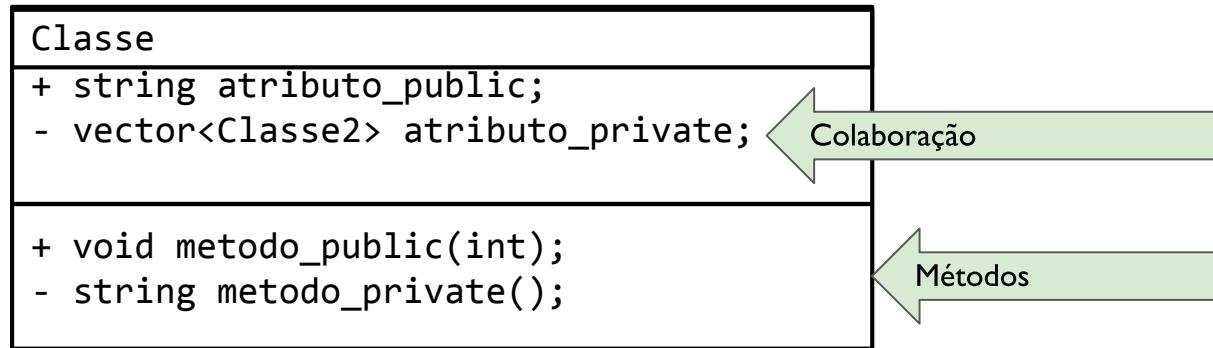
# Diagramas de Classe

- Ajudam a definir as responsabilidades e as colaborações



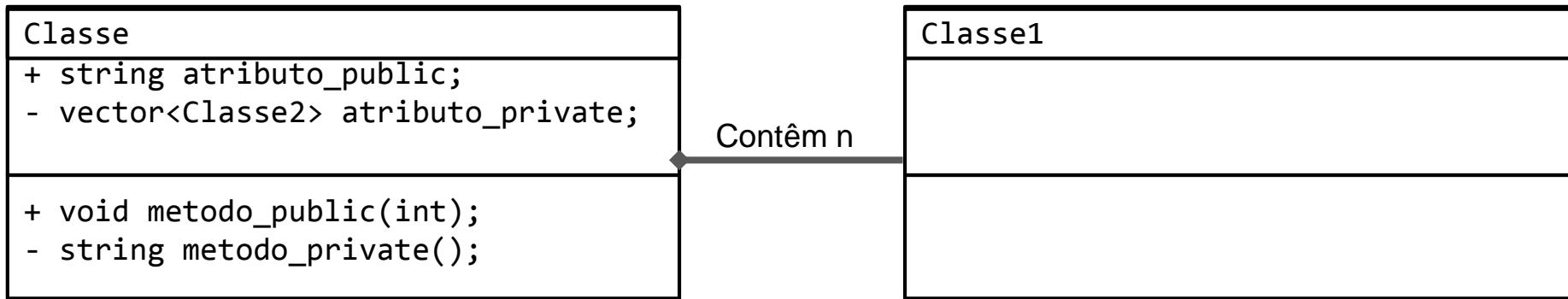
# Diagramas de Classe

- Ajudam a definir as responsabilidades e as colaborações

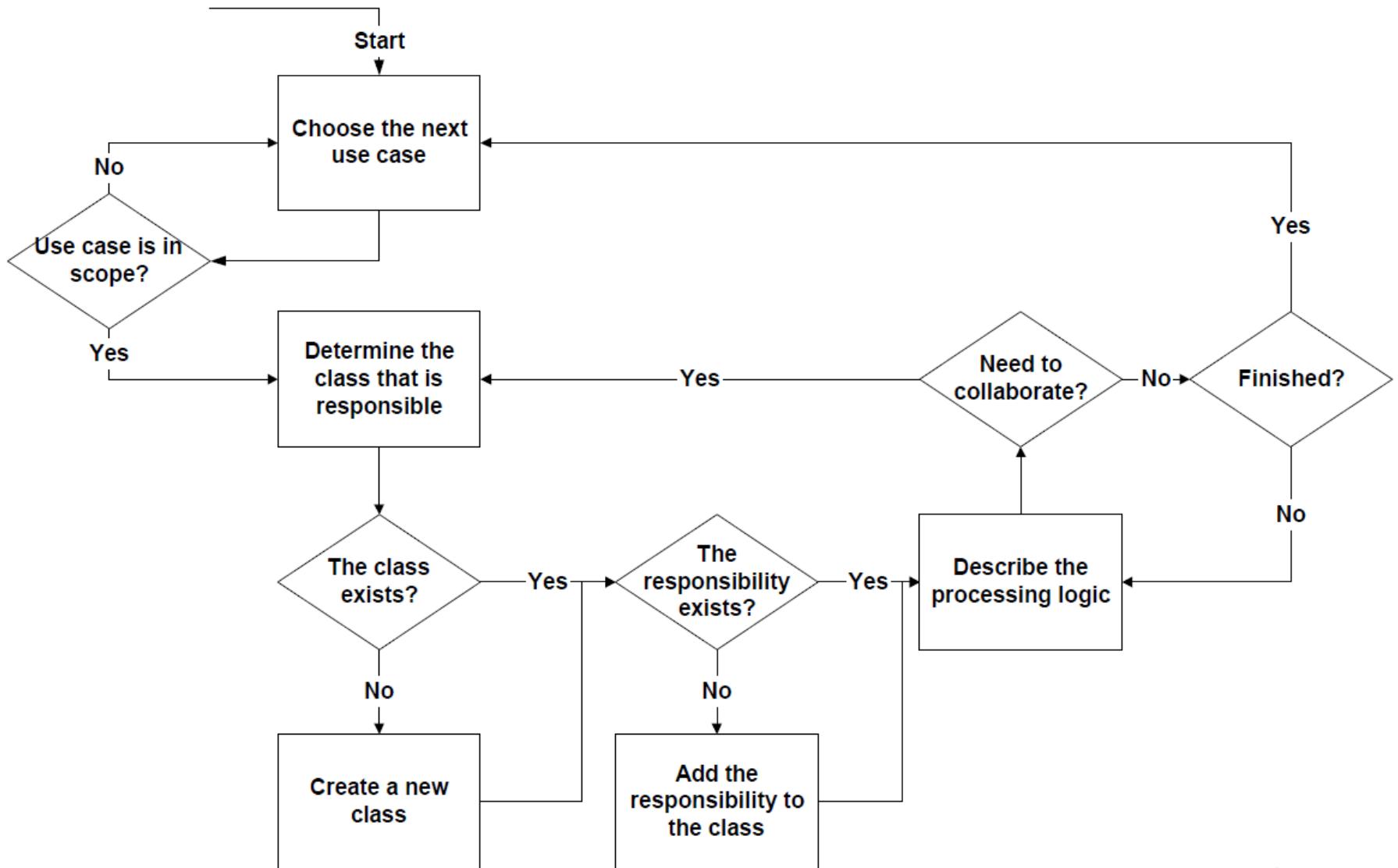


# Diagramas de Classe

- Ajudam a definir as responsabilidades e as colaborações



# Sessões de Modelagem



# Modelagem

- Vantagens**
  - Maior conhecimento do domínio do problema
  - Ótimo ambiente para aprendizado
- Desvantagens**
  - Aplicação limitada em relação ao design
  - Pouco formal, ainda distante da implementação, ...
- Dificuldades em juntar toda a equipe**

# **Baixo Acoplamento e Alta Coesão**

---

# Minimizar dependências

- Como minimizar as dependências e, ao mesmo tempo, maximizar o reuso?

# Definição

- Como minimizar as dependências e, ao mesmo tempo, maximizar o reuso?
- Acoplamento mensura:
  - Quão conectada é uma classe
  - Quanto ela possui conhecimento de outra

# Forte Acoplamento

1. Mudança às classes correlatas
  - a. Impactam na classe acoplada
2. Difícil entender o comportamento
  - a. Comportamento depende de outras classes
3. Difícil re-utilizar
  - a. Muita bagagem

# Mensurando Acoplamento

- X tem um atributo que referencia uma instância de Y
- X tem um método que referencia uma instância de Y
  - Pode ser parâmetro, variável local, objeto retornado pelo método
- Próximas aulas:
  - X é uma subclasse direta ou indireta de Y
  - X implementa a interface Y

# Acoplamento

- Tem que existir
- Porém queremos minimizar o mesmo

# Problema

- Lista de Alunos Ordenados por Matrícula

# Solução 0

## Quais são os problemas?

```
#ifndef LISTA_ALUNO_PDS2_H
#define LISTA_ALUNO_PDS2_H

#include <vector>
#include "aluno.h"

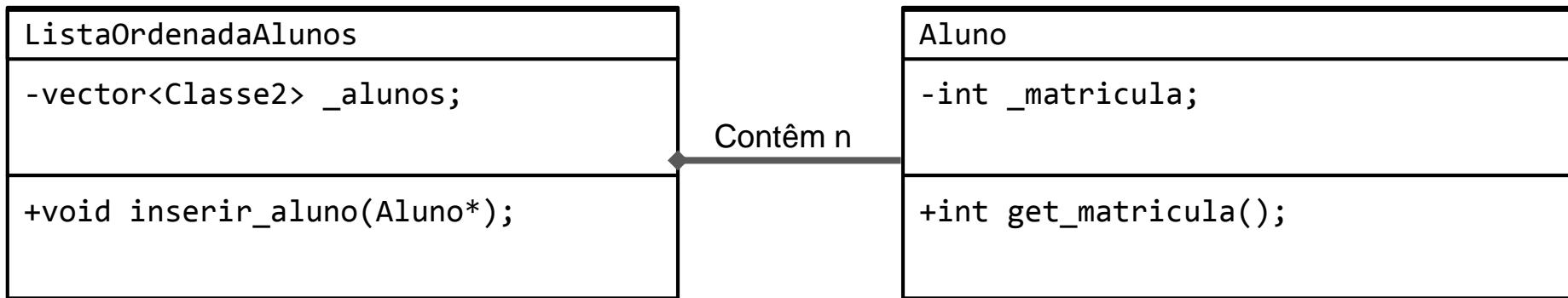
class ListaOrdenadaAlunos {
private:
    std::vector<Aluno *> _alunos;
public:
    void inserir_aluno(Aluno *aluno);
};

#endif
```

# Solução 0

Em UML

- Parece que temos um acoplamento baixo



# Solução 0

## Quais são os problemas?

- A lista funciona para um tipo apenas
  - Alto acoplamento
- Nem sempre é sobre o agora
  - Pensar no futuro

```
#include "listaaluno.h"

void ListaOrdenadaAlunos::inserir_aluno(Aluno *aluno) {
    auto it = this->_alunos.begin();
    auto ed = this->_alunos.end();
    while (it != ed && (*it)->get_matricula() < aluno->get_matricula()) {
        it++;
    }
    this->_alunos.insert(it, aluno);
}
```

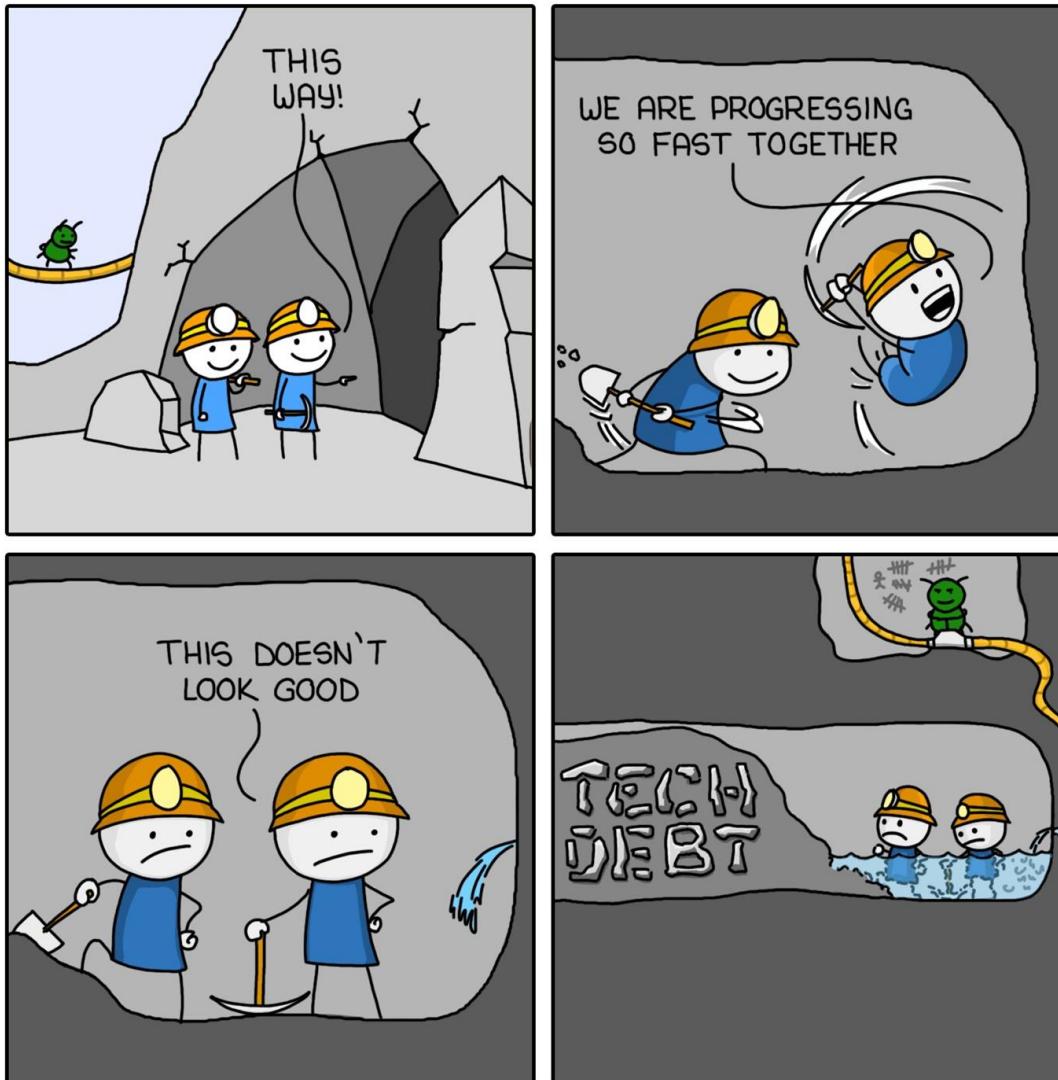
# Pensando no futuro

Em UML

- Uma lista ordenada de disciplinas?!
- Ordenar Alunos pelo nome?
- Ordenar Disciplinas pelo nome?
- Temos um tipo muito específico
  - Acoplamento nem sempre é contar arestas em um código UML

# Technical Debt

## TECH DEBT



MONKEYUSER.COM

# Solução Boa

```
comparator ComparaAlunoMatricula
```

```
+bool operator()(Aluno*, Aluno*);
```

```
comparator ComparaDisciplinaNome
```

```
+bool operator()(Disc*, Disc*);
```

```
Aluno
```

```
-int _matricula;
```

```
+int get_matricula();
```

# Usando o Comparator

- Podemos reutilizar
- Basta combinar com um set
  - Elementos ordenados
- Podemos usar em outros contextos

```
struct aluno_comparator_f {  
    bool operator()(const Aluno &aluno1, const Aluno &aluno2) const {  
        return aluno1.get_matricula() < aluno2.get_matricula();  
    }  
};
```

# Coesão

- A coesão mede quão relacionados ou focados estão as responsabilidades da classe
- Uma maior coesão:
  - Classes com propósitos bem definidos
- Com baixa coesão:
  - God Classes

# Software de Álgebra Linear

- Precisamos representar:
  - Vetores
  - Vetores esparsos
  - Matrizes
  - Matrizes esparsas
- Cada um com operações
- Salvar dados no disco

# Classe Muito Grande

DataClass

```
-double **_dados;  
-int _ndim;  
-bool _esparsa;  
  
+double media();  
+double mediana();  
+double desvio_padrao();  
+double produto_matricial(DataClass);  
+double produto_interno(DataClass);  
+double produto_matricial_sparse(DataClass);  
+double produto_interno_sparse(DataClass);  
+DataClass ordenar(DataClass);  
+void salvar_disco(std::string arquivo);  
+void carregar_dados_do_disco(std::string arquivo);  
. . .
```

# Nova Solução

## Classes com responsabilidades bem definidas

Vetor

```
-double *_dados;
```

operações vetores

VetorEsparsa

```
-std::map<int, double>;
```

operações vetor esparsa

Matriz

```
-double **_dados;
```

operações matrizes

MatrizEsparsa

```
-std::map<pair, double>;
```

operações matrizes

# Por fim

- Lembre-se dos conceitos desta aula
- Acoplamento
- Coesão
- Vamos re-lembrar do mesmo ao discutir herança e polimorfismo

# Programação e Desenvolvimento de Software 2

## Herança e Composição

---

Flavio Figueiredo

<http://github.com/flaviovdf/programacao-2>

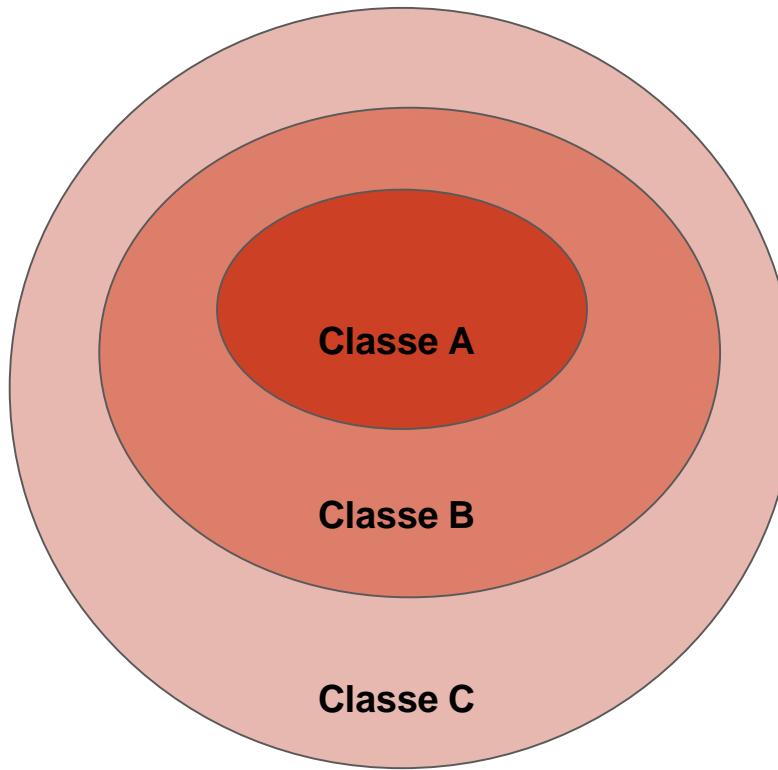
# Introdução

- Técnica para reutilizar características de uma classe na definição de outra classe
- Hierarquia de classes
- Terminologias relacionadas à Herança
  - Classes mais genéricas: superclasses (pai)
  - Classes especializadas: subclasses (filha)

# Introdução

- Superclasses
  - Devem guardar membros em comum
- Subclasses
  - Acrescentam novos membros (especializam)
- Componentes facilmente reutilizáveis
  - Facilita a extensibilidade do sistema

## Contexto de Classe



# Herança

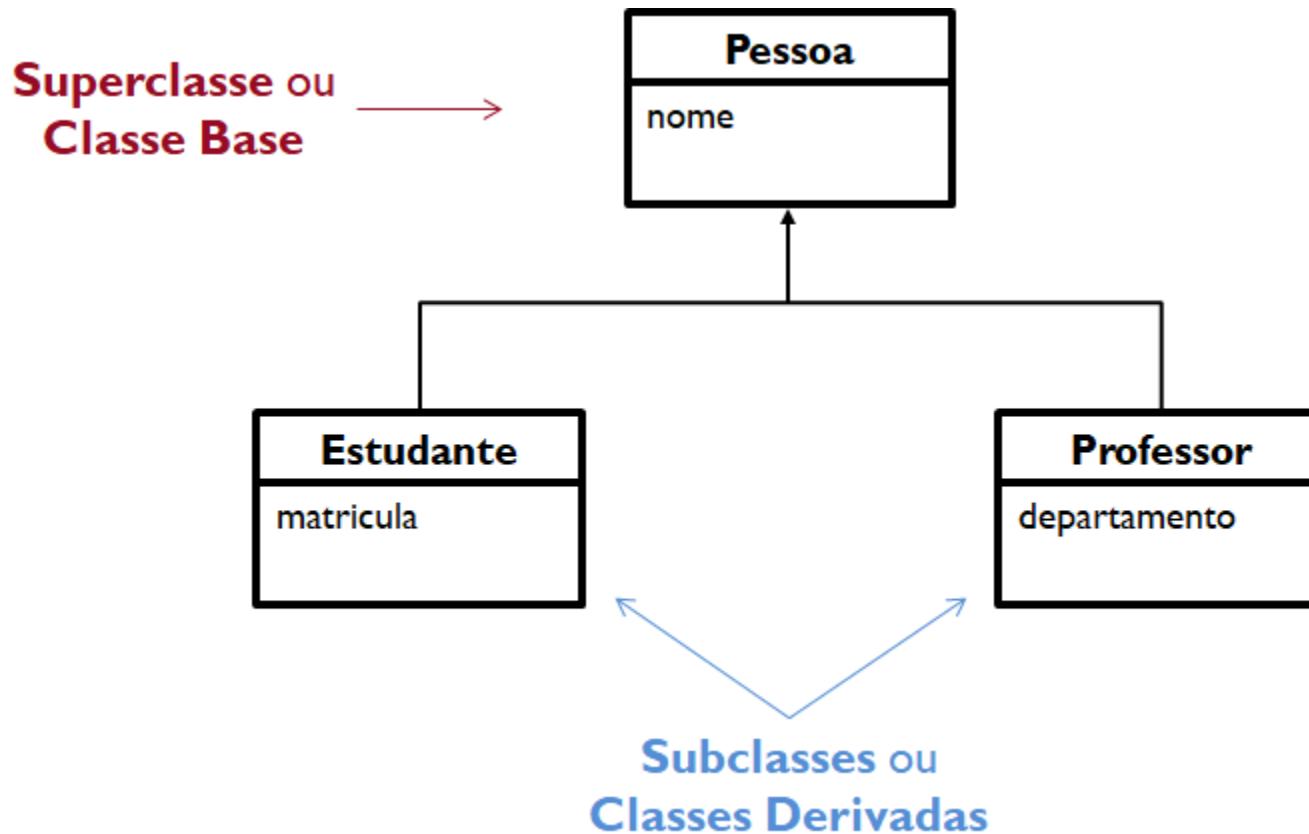
- Os atributos e métodos são herdados por todos os objetos dos níveis mais baixos
  - Considerando o modificador de acesso
- Diferentes subclasses podem herdar as características de uma ou mais superclasses
  - Herança simples
  - Herança múltipla (evitar)

# Herança

## Benefícios

- Reutilização de código
  - Compartilhar similaridades
  - Preservar as diferenças
- Facilita a manutenção do sistema
  - Maior legibilidade do código existente
  - Quantidade menor de linhas de código
  - Alterações em poucas partes do código

# Herança simples



# Herança Simples

```
#ifndef PDS2_PESSOA_H
#define PDS2_PESSOA_H

#include <string>

class Pessoa {
private:
    const std::string _nome;
public:
    Pessoa(std::string nome);
    virtual std::string get_nome() const;
};

#endif
```

```
#ifndef PDS2_ESTUDANTE_H
#define PDS2_ESTUDANTE_H

#include "pessoa.h"

class Estudante : public Pessoa {
private:
    const int _matricula;
public:
    Estudante(std::string nome,
              int matricula);
    int get_matricula() const;
};

#endif
```

# Todo Estudante é uma Pessoa

```
#ifndef PDS2_PESSOA_H
#define PDS2_PESSOA_H

#include <string>

class Pessoa {
private:
    const std::string _nome;
public:
    Pessoa(std::string nome);
    virtual std::string get_nome() const;
};

#endif
```

```
#ifndef PDS2_ESTUDANTE_
#define PDS2_ESTUDANTE_

#include "pessoa.h"

class Estudante : public Pessoa {
private:
    const int _matricula;
public:
    Estudante(std::string nome,
              int matricula);
    int get_matricula() const;
};

#endif
```

# Uso

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.get_nome() << std::endl;
    std::cout << "O estudante é: " << estudante.get_nome() << std::endl;

    return 0;
}
```

# Note o uso de `get_nome` nos dois tipos

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.get_nome() << std::endl;
    std::cout << "O estudante é: " << estudante.get_nome() << std::endl;

    return 0;
}
```

# Implementação

- Pessoa é uma classe normal

```
#include "pessoa.h"

Pessoa::Pessoa(std::string nome):
    _nome(nome) {}

std::string Pessoa::get_nome() const {
    return this->_nome;
}
```

# Implementação

## □ Note o construtor diferente (initializer)

```
#include "pessoa.h"

Pessoa::Pessoa(std::string nome) :  
    _nome(nome) {}  
  
std::string Pessoa::get_nome() const {  
    return this->_nome;  
}
```

Construtor esquisito (vimos rapidamente)

# Implementação

- Vamos falar do mesmo em breve

```
#include "pessoa.h"

Pessoa::Pessoa(std::string nome) : _nome(nome) {}

std::string Pessoa::get_nome() const {
    return this->_nome;
}
```

Construtor esquisito (vimos rapidamente)

# Estudante

- Novamente uma classe quase normal
- Porém sem `get_nome`
- Construtor esquisito novamente

```
#include "estudante.h"

Estudante::Estudante(std::string nome, int matricula):
    Pessoa(nome), _matricula(matricula) {}

int Estudante::get_matricula() const {
    return this->_matricula;
}
```

# Herança

- Todo Estudante é uma Pessoa
- Então:
  - Todo estudante tem um nome
  - Além de um método `get_nome`

# Herança

- Não implementamos `get_nome` abaixo
- Foi herdado de `Pessoa`

```
#include "estudante.h"

Estudante::Estudante(std::string nome, int matricula):
    Pessoa(nome), _matricula(matricula) {}

int Estudante::get_matricula() const {
    return this->_matricula;
}
```

# Construtor "Initializer List"

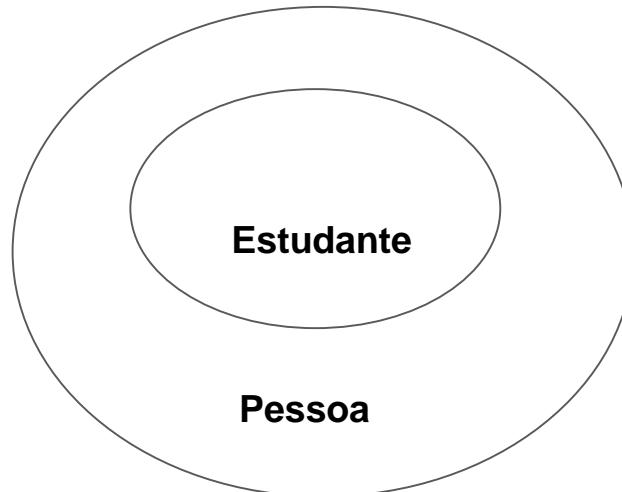
- Garante que toda memória é inicializada

```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```

# Construtor "Initializer List"

- Todo estudante é uma pessoa

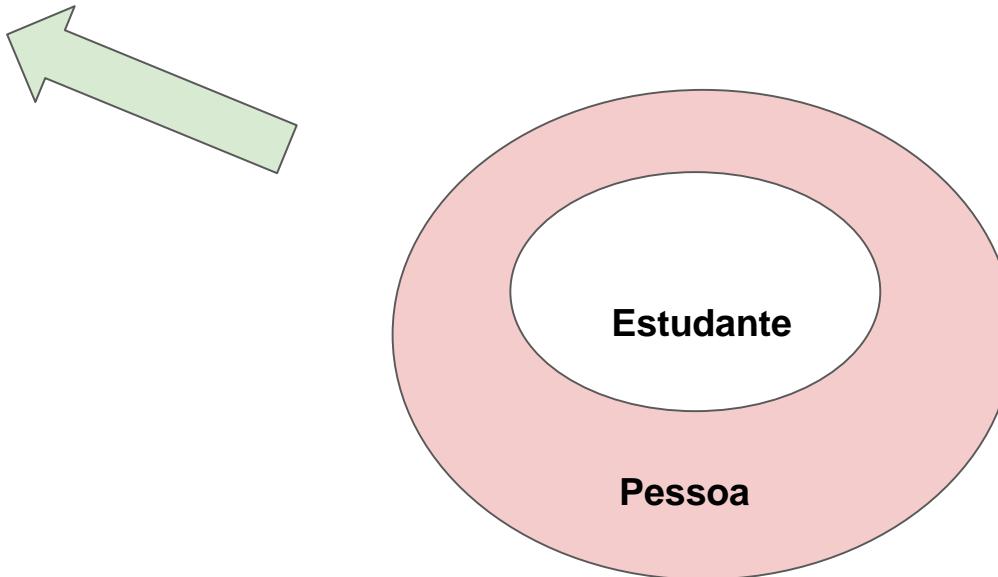
```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```



# Construtor "Initializer List"

- Temos que iniciar a memória da Pessoa

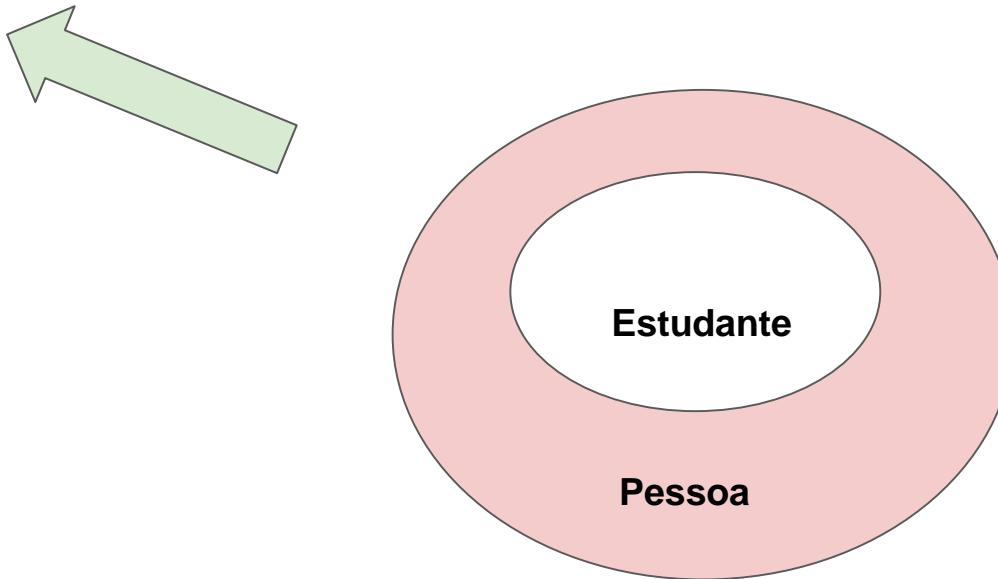
```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```



# Construtor "Initializer List"

- Ou seja, setar o nome nesse caso

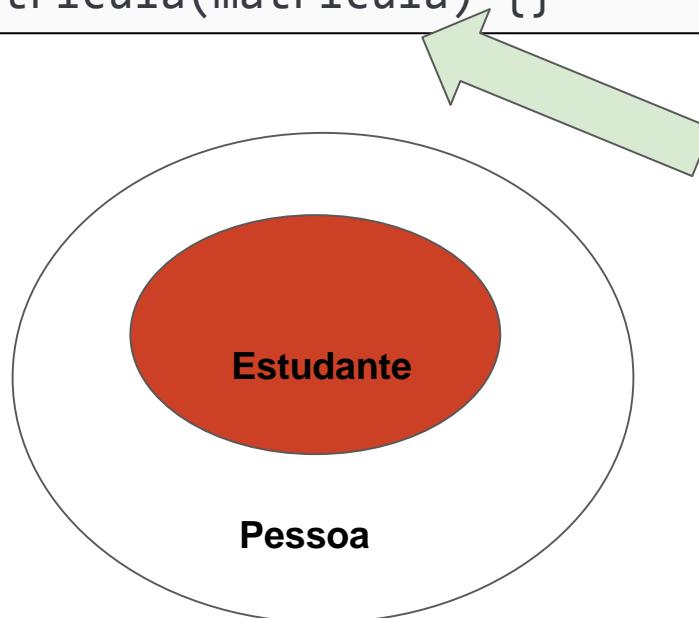
```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```



# Construtor "Initializer List"

## □ Depois do Estudante

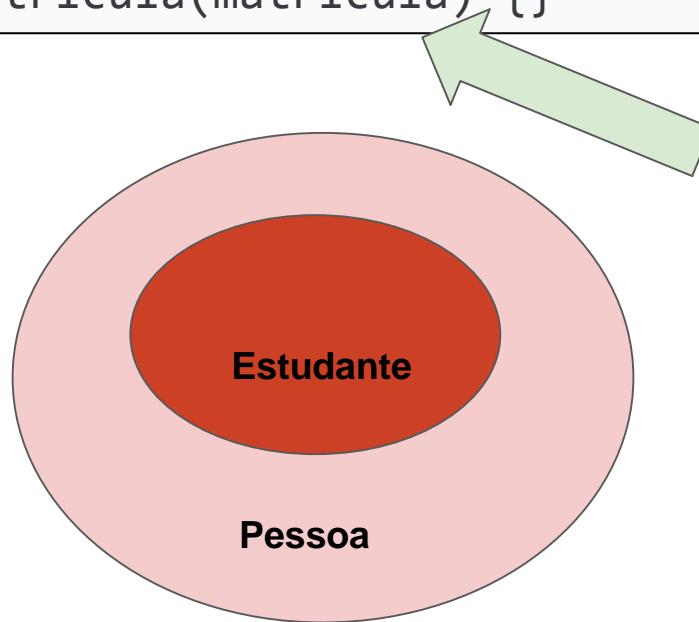
```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```



# Construtor "Initializer List"

## □ Setar o campo matricula

```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```



# Erro de compilação

- ☐ Não iniciamos a parte pessoa

```
Estudante::Estudante(std::string nome, int matricula) {  
    this->_matricula = matricula;  
}
```



# (Entendendo) Initializer List

- Lembrando C++ a linha abaixo chama um construtor:

```
Pessoa p("Flavio F.");
```

- ▣ Atalho para:

```
Pessoa p = Pessoa("Flavio F.");
```

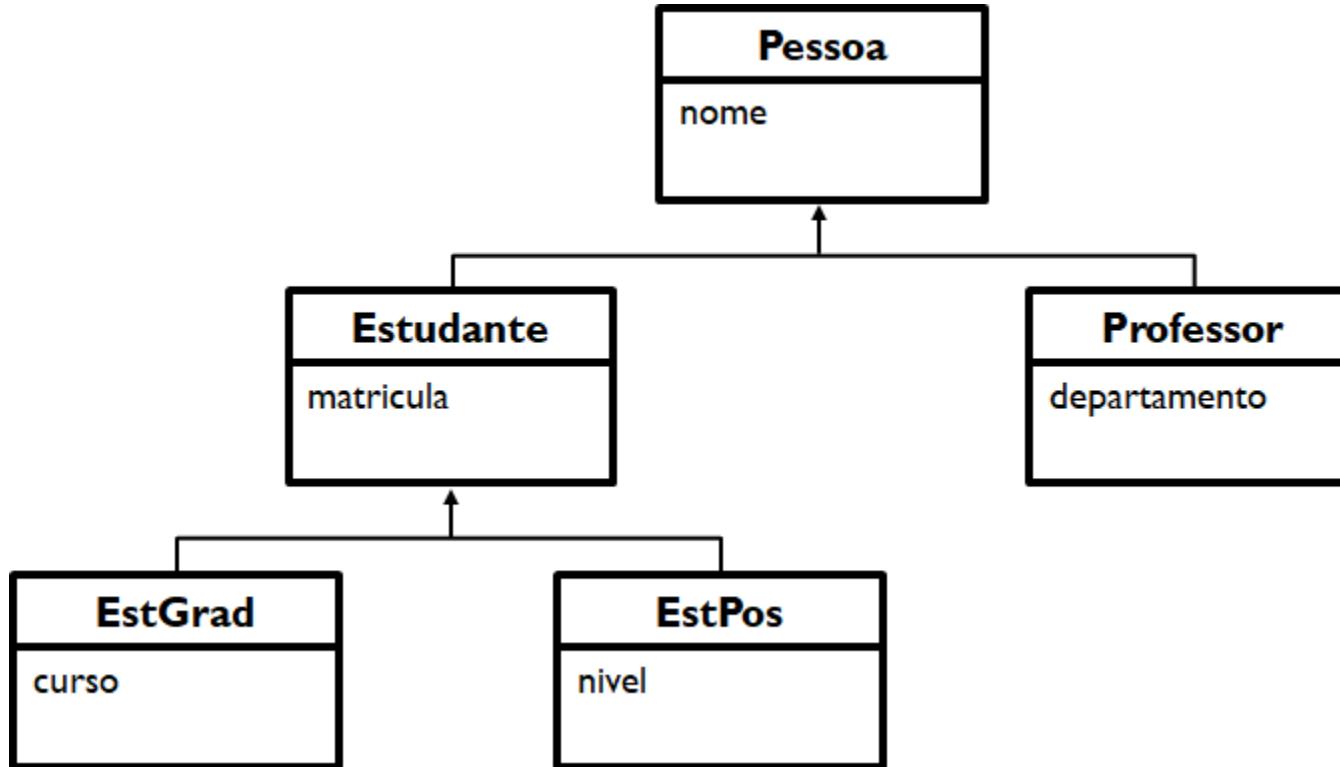
# (Entendendo)Initializer List

- Aqui é a mesma coisa:
  - Construa a Pessoa antes do Estudante
  - depois
  - Construa a matrícula
- this->\_matricula = matricula;**

```
Estudante::Estudante(std::string nome, int matricula):  
    Pessoa(nome), _matricula(matricula) {}
```

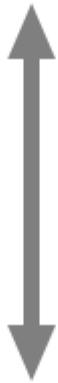


# Herança simples em vários níveis

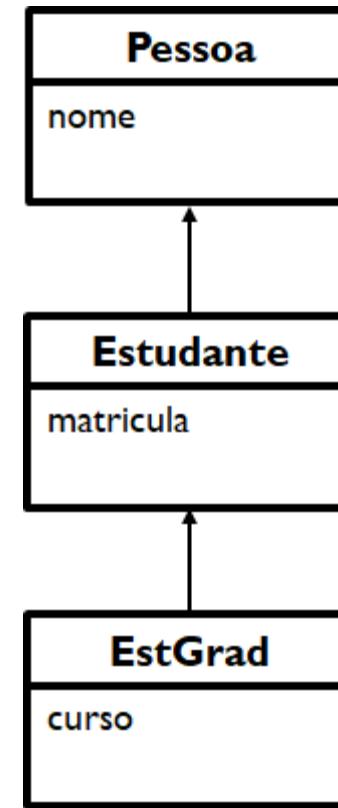


# Herança simples

**Generalização**



**Especialização**



# Herança simples

## Sobrescrita de métodos

- Métodos são sobrescritos (**overriding**)
  - Diferente de sobrecarga!
  - Mesma assinatura e tipo de retorno (!)
  - Métodos private não são sobrescritos

# Herança simples

## Sobrescrita de métodos

- Métodos sobre escritos devem ser ‘virtuais’
- Atributos não são re-definíveis
  - Se atributo de mesmo nome for definido na subclasse, a definição na superclasse é ocultada
- Membros estáticos
  - Não são redefinidos, mas ocultados
  - Como o acesso é feito pelo nome da classe, estar ou não ocultado terá pouco efeito

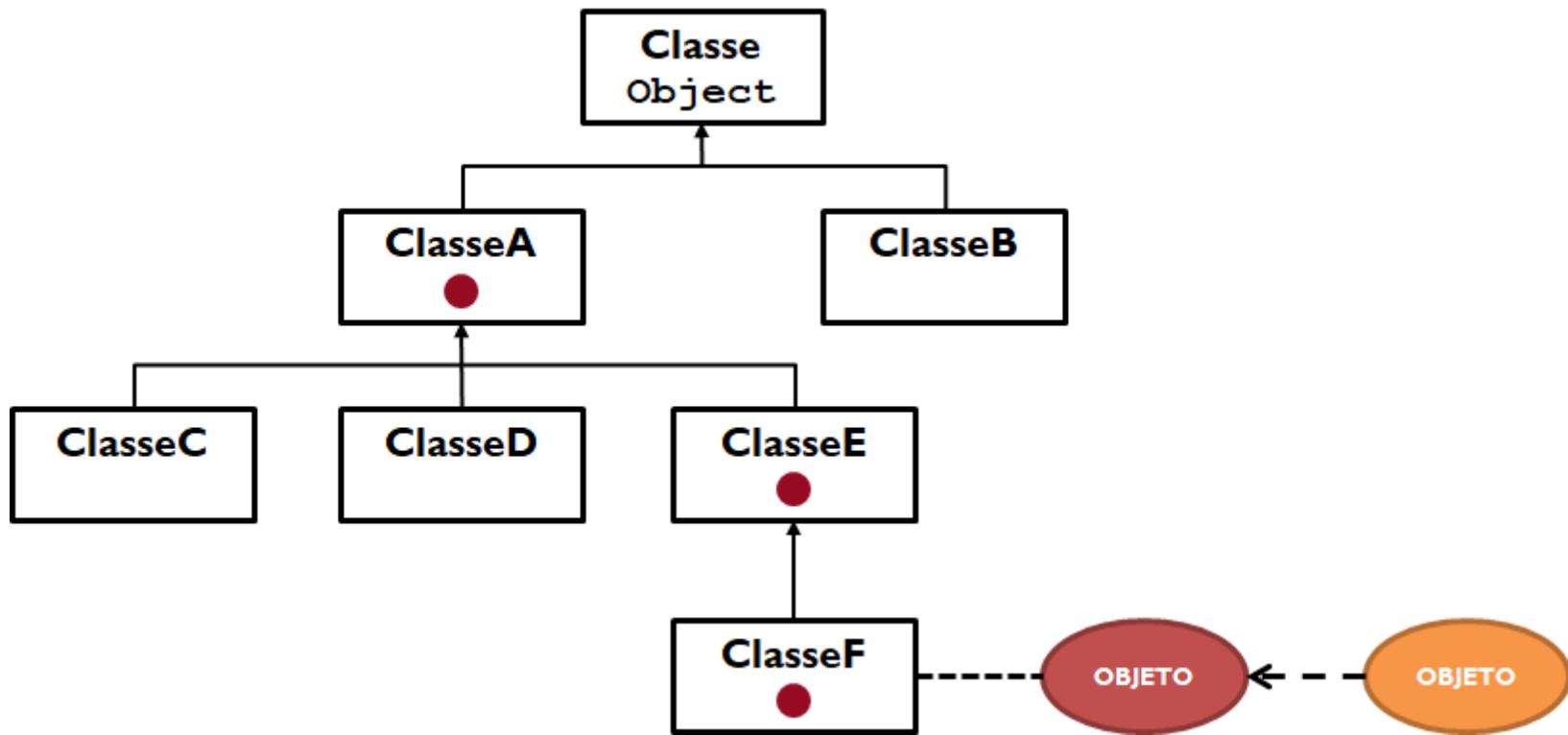
# Herança simples

## Sobrescrita de métodos

- Métodos sobreescritos devem ser ‘virtuais’
- Lembre-se:
  - nomes iguais não definem uma conexão

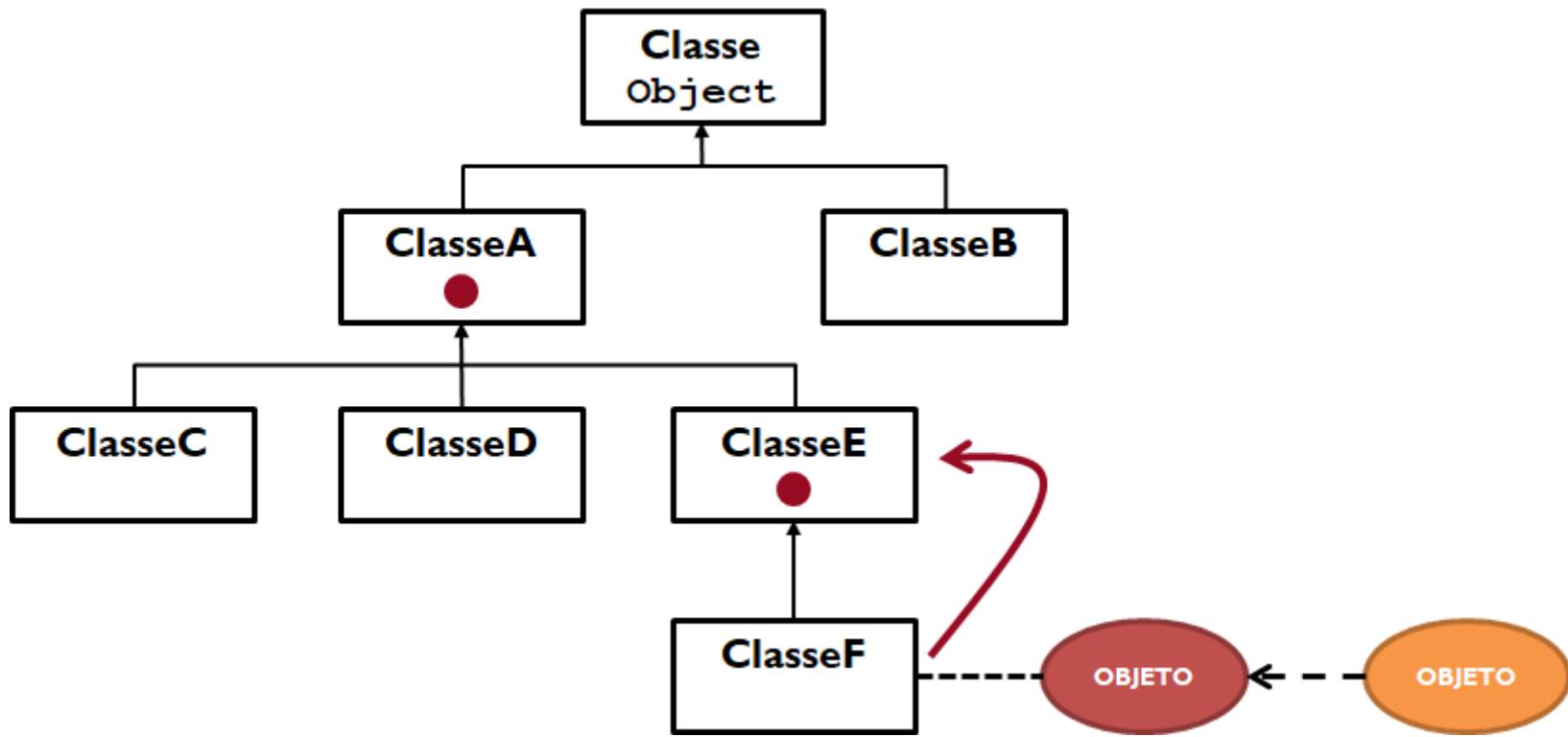
# Herança simples

Sobrescrita de métodos com **virtual**



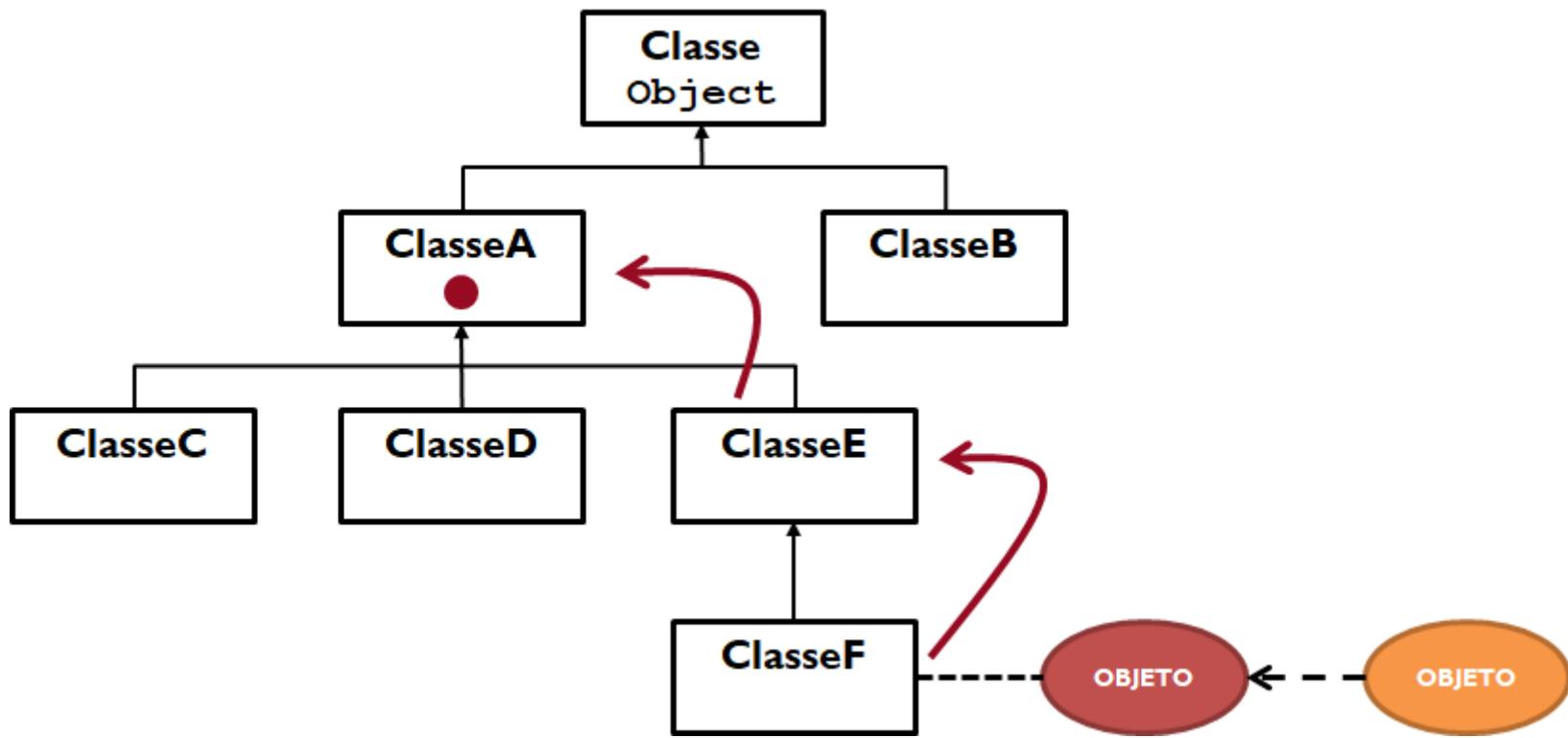
# Herança simples

Sobrescrita de métodos com **virtual**



# Herança simples

Sobrescrita de métodos com **virtual**



# Definindo uma Sobrecarga

Dois métodos com o mesmo nome usando `virtual`

```
#ifndef PDS2_PESSOA_H
#define PDS2_PESSOA_H

#include <string>

class Pessoa {
private:
    const std::string _nome;
public:
    Pessoa(std::string nome);
    virtual std::string defina_meu_tipo() const;
};

#endif
```

# Definindo uma Sobrecarga

Dois métodos com o mesmo nome usando `virtual`

```
#ifndef PDS2_ESTUDANTE_H
#define PDS2_ESTUDANTE_H

#include "pessoa.h"

class Estudante : public Pessoa {
private:
    const int _matricula;
public:
    Estudante(std::string nome,
              int matricula);
    int get_matricula() const;
    virtual std::string defina_meu_tipo() const override;
};

#endif
```

# Override

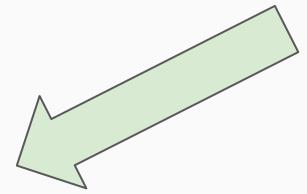
Ajuda o compilador

```
#ifndef PDS2_ESTUDANTE_H
#define PDS2_ESTUDANTE_H

#include "pessoa.h"

class Estudante : public Pessoa {
private:
    const int _matricula;
public:
    Estudante(std::string nome,
              int matricula);
    int get_matricula() const;
    virtual std::string defina_meu_tipo() const override;
};

#endif
```



# Sobrescrita

## Mudando os .cpp (focando no método novo)

```
#include "pessoa.h"
Pessoa::Pessoa(std::string nome):
    _nome(nome) {}

std::string Pessoa::defina_meu_tipo() const {
    return "Sou uma pessoa!";
}
```

```
#include "estudante.h"

Estudante::Estudante(std::string nome, int matricula):
    Pessoa(nome), _matricula(matricula) {}

std::string Estudante::defina_meu_tipo() const override {
    return "Sou um estudante";
}
```

# Exemplo de uso

## Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```

# Exemplo de uso

## Qual a saída abaixo?!

```
$ ./main
```

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```



# Exemplo de uso

## Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```

```
$ ./main
A pessoa é: Sou uma pessoa
```

# Exemplo de uso

## Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```

```
$ ./main
A pessoa é: Sou uma pessoa
O estudante é: Sou um estudante
```

# Exemplo de uso

## Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```

```
$ ./main
A pessoa é: Sou uma pessoa
O estudante é: Sou um estudante
```

# Exemplo de uso

## Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```



```
$ ./main
A pessoa é: Sou uma pessoa
O estudante é: Sou um estudante
Na função: Sou uma pessoa
```

# Exemplo de uso

## Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```

```
$ ./main
A pessoa é: Sou uma pessoa
O estudante é: Sou um estudante
Na função: Sou uma pessoa
Na função: Sou uma estudante
```

# Polimorfismo

- Este é um exemplo de polimorfismo
- Comportamento diferente para uma mesma chamada
- Definida em tempo de execução
- Vamos explorar melhor no futuro

# Fonte de bugs

Dois métodos com o mesmo nome sem virtual

```
#ifndef PDS2_PESSOA_H
#define PDS2_PESSOA_H

#include <string>

class Pessoa {
private:
    const std::string _nome;
public:
    Pessoa(std::string nome);
    std::string definia_meu_tipo() const;
};

#endif
```

```
#ifndef PDS2_ESTUDANTE_H
#define PDS2_ESTUDANTE_H

#include "pessoa.h"

class Estudante : public Pessoa {
private:
    const int _matricula;
public:
    Estudante(std::string nome,
              int matricula);
    int get_matricula() const;
    std::string definia_meu_tipo() const;
};

#endif
```

# Fonte de bugs

## Mudando os .cpp (focando no método novo)

```
#include "pessoa.h"
Pessoa::Pessoa(std::string nome):
    _nome(nome) {}

std::string Pessoa::defina_meu_tipo() const {
    return "Sou uma pessoa!";
}
```

```
#include "estudante.h"

Estudante::Estudante(std::string nome, int matricula):
    Pessoa(nome), _matricula(matricula) {}

std::string Estudante::defina_meu_tipo() const {
    return "Sou um estudante";
}
```

# Fonte de bugs

Qual a saída abaixo?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```

# Fonte de bugs

Esquisito.

```
$ ./main
A pessoa é: Sou uma pessoa!
O estudante é: Sou um estudante
Na função: Sou uma pessoa!
Na função: Sou uma pessoa!
```

# Early Binding

Em tempo de compilação

- Sem **virtual** o compilador usa o tipo **mais próximo**. Na função é **Pessoa**.

```
void f(Pessoa &pessoa) {  
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;  
}
```

```
$ ./main  
A pessoa é: Sou uma pessoa!  
O estudante é: Sou um estudante  
Na função: Sou uma pessoa!  
Na função: Sou uma pessoa!
```



# Virtual Override

Não é uma banda de Metal

- Virtual → Late Binding
  - Em tempo de execução
- Override → Indica que estamos realizando uma sobrescrita
  - Não é um novo método **virtual**
  - É a sobrescrita da superclasse
  - Não é necessário
    - Evita bugs logo em tempo de compilação

# Late Binding

Em tempo de execução

- O **virtual** faz o tipo ser definido em tempo de execução. Ou seja, **Estudante**.

```
void f(Pessoa &pessoa) {  
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;  
}
```

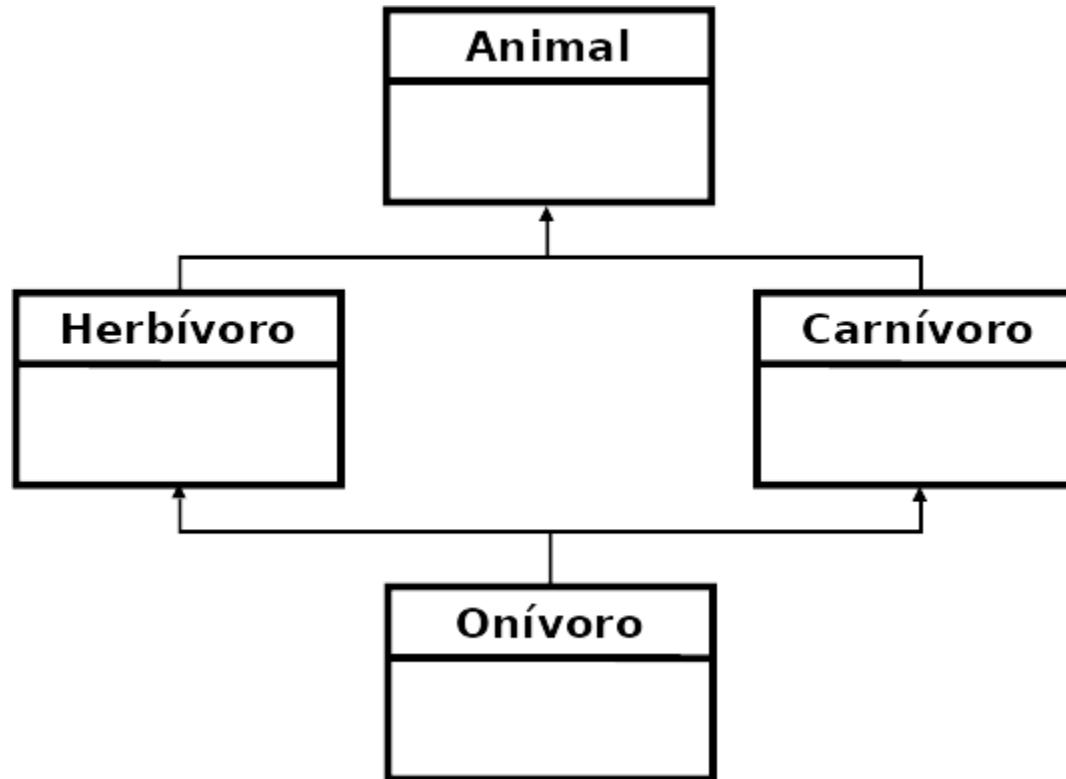
```
$ ./main  
A pessoa é: Sou um estudante  
O estudante é: Sou um estudante  
Na função: Sou uma pessoa!  
Na função: Sou um estudante
```



# Herança múltipla

- Subclasse herda de mais de uma superclasse
  - Nem todas as linguagens permitem isso
- Problemas
  - Dificulta a manutenção do sistema
  - Também dificulta o entendimento
  - Reduz a modularização (super objetos)
    - Classes que herdam de todo mundo
    - Saída do preguiçoso

# Herança múltipla



# Herança múltipla

- Possível em C++
- Nunca use.

```
class Onivoro : public Herbivoro, public Carnivoro {  
};
```

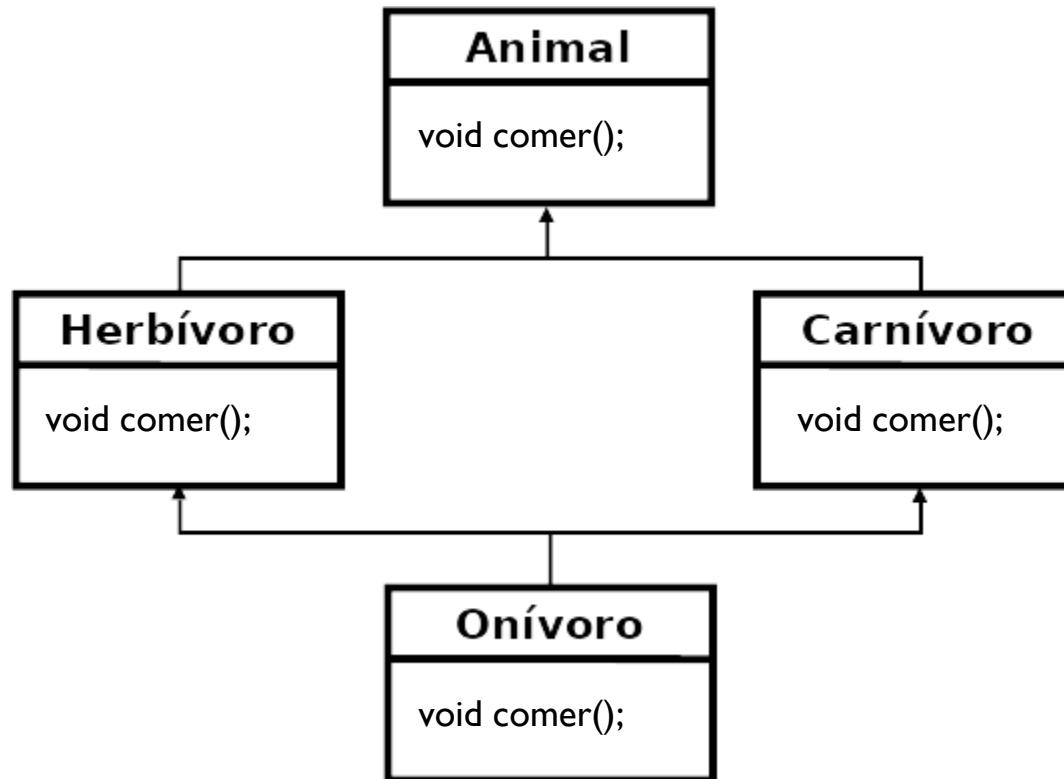
# Herança múltipla

- Possível em C++
- Nunca use.
- Sério.

```
class Onivoro : public Herbivoro, public Carnivoro {  
};
```

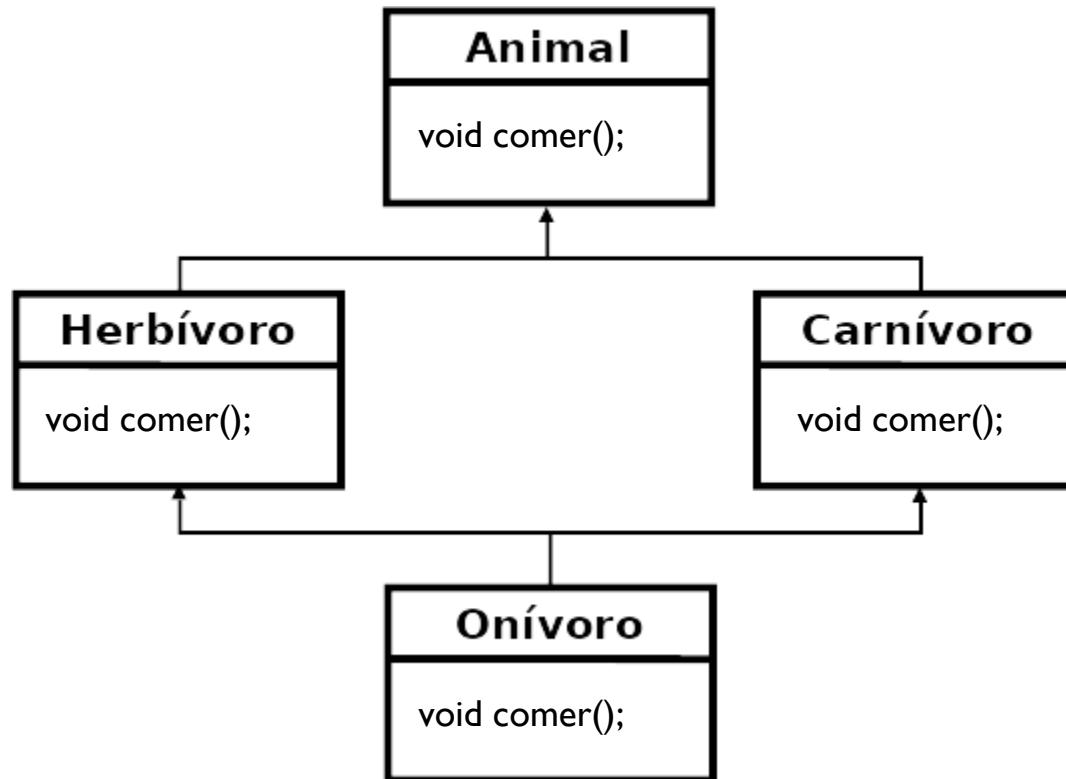
# Herança múltipla

Existem 4 definições de comer



# Herança múltipla

Qual vai ser executada?



# Herança

## Críticas

- “Fere” o princípio do encapsulamento
  - Membros fazem parte de várias classes
- Cria interdependência entre classes
  - Mudanças em superclasses podem ser difíceis
- Como resolver isso?

# Herança

## Críticas

- “Fere” o princípio do encapsulamento
  - Membros fazem parte de várias classes
- Cria interdependência entre classes
  - Mudanças em superclasses podem ser difíceis
- Como resolver isso?

Composition is often more appropriate than inheritance.  
When using inheritance, make it public.

– Google C++ Style Guide

# Herança vs. Composição

## □ Herança

- Relação do tipo “é um” (is-a)
- Subclasse tratada como a superclasse
- Estudante é uma Pessoa

## □ Composição

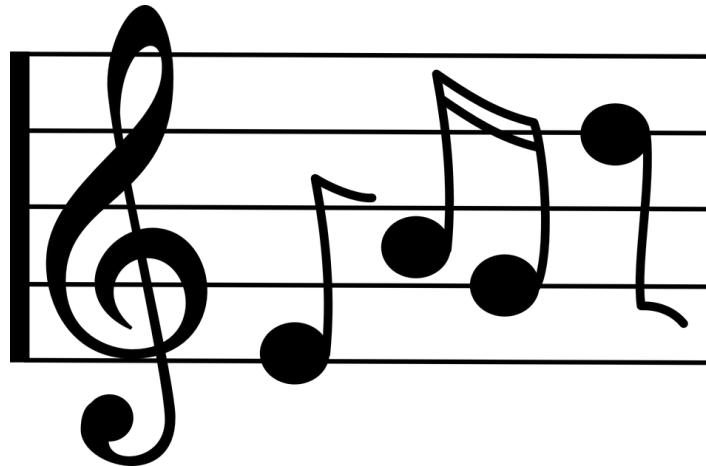
- Relação do tipo “tem um” (has-a)
- Objeto possui objetos ( $\geq 1$ ) de outras classes
- Estudante tem um Curso

# Composição

- Técnica para criar um novo tipo não pela derivação, mas pela junção de outras classes de menor complexidade
- Não existe palavra-chave ou recurso
- Conceito lógico de agrupamento
  - Modo particular de implementação
- Funciona muito bem com **interfaces** (aulas futuras)

# Composição

- Ao invés de copiar o comportamento
- Repassamos a responsabilidade
  - Boa prática!
- Cada objeto faz uma única coisa
  - Compomos os mesmos



# Composição

- Ao invés de copiar o comportamento
- Repassamos a responsabilidade
  - Boa prática!
- Antes de usar herança pense:
  - (1) faz sentido a relação de é (is-a)?
  - (2) a composição fica mais complicado?
- Se qualquer um dos dois for **não**
  - Não use herança.

# Programação e Desenvolvimento de Software 2

## Herança (Parte 2)

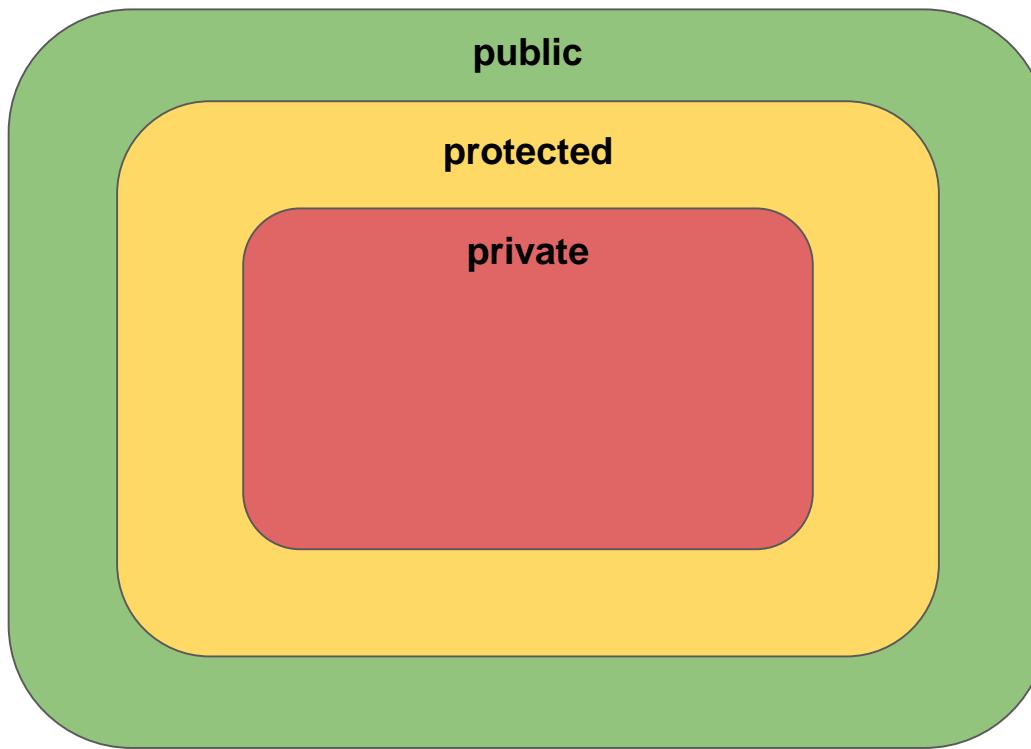
---

Flavio Figueiredo

<http://github.com/flaviovdf>

# Encapsulamento

## Modificadores de acesso



# Protected

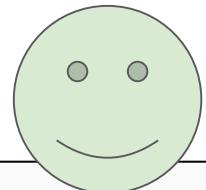
- Explora a Hierarquia de Classes
- Apenas subclasses podem acessar
- Não é visível para fora

# Protected

```
class Base {  
protected:  
    int i = 0;  
};  
  
class Derived : public Base {  
public:  
    int f() {  
        i++;  
        return i;  
    }  
};
```

```
int main() {  
    Derived d1;  
    std::cout << d1.f() << std::endl;  
  
    Derived d2;  
    std::cout << d2.f() << std::endl;  
    return 0;  
}
```

# Exemplo de protected



```
#include <string>

class Carta {
protected:
    int _dano;
public:
    Carta(int dano, std::string nome);
    virtual int get_dano();
    std::string get_nome();
};

int Carta::get_dano() {
    return this->_dano;
}
```

```
class CartaBoost : public Carta {
private:
    int _boost;
    std::string _nome;
public:
    CartaBoost(int boost, int dano,
               std::string nome);
    virtual int get_dano() override;
};

int CartaBoost::get_dano() {
    return Carta::_dano + _boost;
}
```

# Exemplo de herança protected



```
#include <string>

class Carta {
private:
    int _dano;
public:
    Carta(int dano, std::string nome);
    virtual int get_dano();
    std::string get_nome();
};

int Carta::get_dano() {
    return this->_dano;
}
```

```
class CartaBoost : public Carta {
private:
    int _boost;
    std::string _nome;
public:
    CartaBoost(int boost, int dano,
               std::string nome);
    virtual int get_dano() override;
};

int CartaBoost::get_dano() {
    return Carta::_dano + _boost;
}
```

# Dano só é visível dentro de carta



```
#include <string>

class Carta {
private:
    int _dano;
public:
    Carta(int dano, std::string nome);
    virtual int get_dano();
    std::string get_nome();
};

int Carta::get_dano() {
    return this->_dano;
}
```

```
class CartaBoost : public Carta {
private:
    int _boost;
    std::string _nome;
public:
    CartaBoost(int boost, int dano,
               std::string nome);
    virtual int get_dano() override;
};

int CartaBoost::get_dano() {
    return Carta::_dano + _boost;
}
```

# Encapsulamento

## Modificadores de herança

- Note que também definimos um modificador ao realizar a herança

```
class Base {  
protected:  
    int i = 0;  
};  
  
class Derived : public Base {  
public:  
    int f() {  
        i++;  
        return i;  
    }  
};
```

# Encapsulamento

## Modificadores de herança

- Indica **como** será herdado
- Diferente de **quais**. Sempre herdamos tudo que é **public**, **protected**

# Encapsulamento

## Modificadores de herança

- Indica **como** será herdado
- Diferente de **quais**
- Herdando como **public**
  - Caso comum
  - Herdamos todo o comportamento publicamente. Isto é, para o mundo externo somos do mesmo tipo

# Encapsulamento

## Modificadores de herança

- Indica **como** será herdado
- Diferente de **quais**
- Herdando como **private**
  - Raro. Prefira encapsulamento
  - Herdamos todo o comportamento em avisar para o mundo. Isto é, temos um estado similar, mas ninguém sabe qual é

# Exemplo

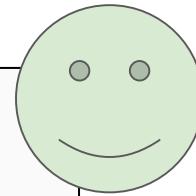
## Herança com Modificador

```
#include <string>

class Carta {
private:
    int _dano;
public:
    Carta(int dano, std::string nome);
    virtual int get_dano();
    std::string get_nome();
};
```

```
class CartaBoost : public Carta {
private:
    int _boost;
    std::string _nome;
public:
    CartaBoost(int boost,
               std::string nome);
    virtual int get_dano() override;
};
```

```
int main() {
    Carta c(2, "Monstro");
    CartaBoost boost(3, "Boost");
    boost.get_nome();
}
```



# Exemplo

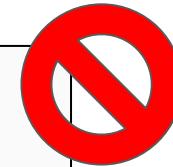
## Herança com Modificador

```
#include <string>

class Carta {
private:
    int _dano;
public:
    Carta(int dano, std::string nome);
    virtual int get_dano();
    std::string get_nome();
};
```

```
class CartaBoost : private Carta {
private:
    int _boost;
    std::string _nome;
public:
    CartaBoost(int boost,
               std::string nome);
    virtual int get_dano() override;
};
```

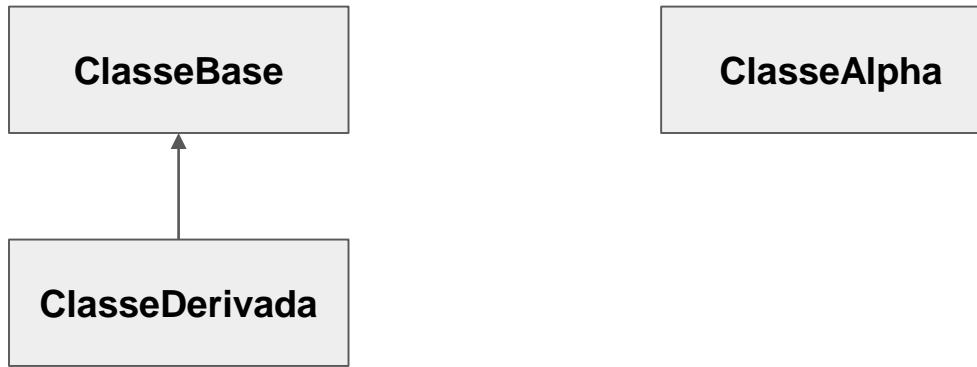
```
int main() {
    Carta c(2, "Monstro");
    CartaBoost boost(3, "Boost");
    boost.get_nome();
}
```



# Encapsulamento

## Modificadores de acesso

- Considerando a visibilidade dos membros da **ClasseBase** de acordo com o modificador de acesso



	ClasseBase	ClasseDerivada	ClasseAlpha
Public	<span style="background-color: green;">■</span>	<span style="background-color: green;">■</span>	<span style="background-color: green;">■</span>
Protected	<span style="background-color: green;">■</span>	<span style="background-color: green;">■</span>	<span style="background-color: darkred;">■</span>
Private	<span style="background-color: green;">■</span>	<span style="background-color: darkred;">■</span>	<span style="background-color: darkred;">■</span>

# Conversão de tipos

- Uma classe, ao herdar de outra, assume o tipo desta onde quer que seja necessário
- Upcasting
  - Conversão para uma classe mais genérica
- Downcasting
  - Conversão para uma classe mais específica

# Conversão de tipos

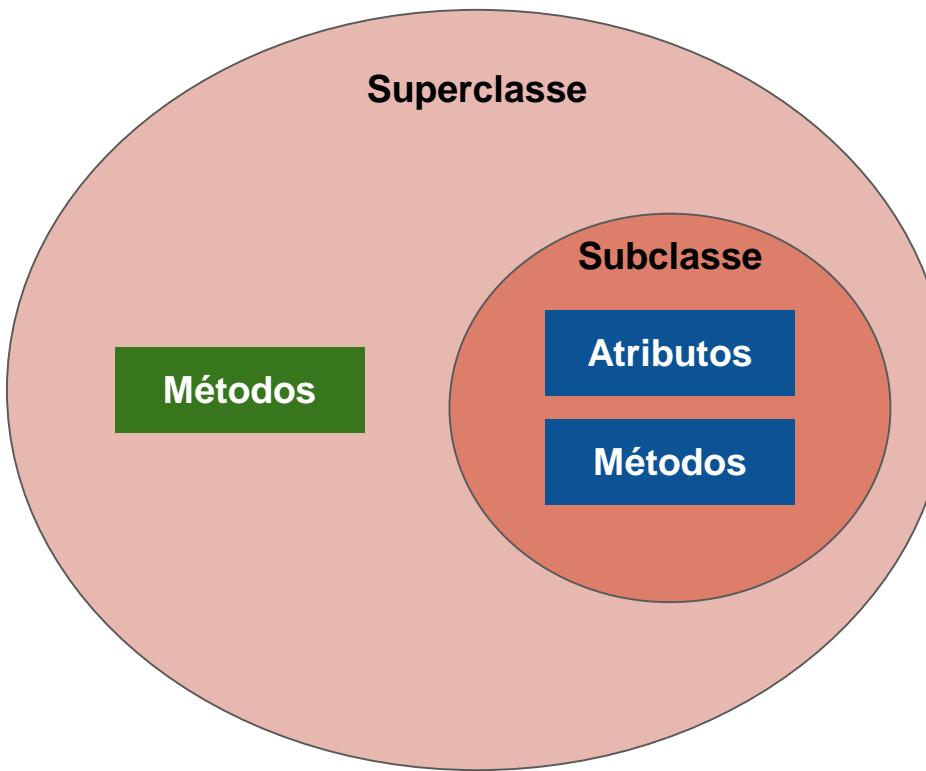
## Upcasting

- Ocorre no sentido Classe  $\Rightarrow$  Superclasse
- Não há necessidade de indicação explícita
- A classe derivada sempre vai manter as características públicas da superclasse

# Conversão de tipos

## Upcasting

### Contexto de Classe



# Upcasting

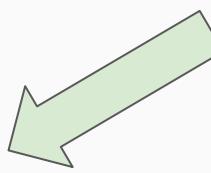
Note que os objetos viram Pessoa auto-magicamente

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

void f(Pessoa &pessoa) {
    std::cout << "Na função: " << pessoa.defina_meu_tipo() << std::endl;
}

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);
    std::cout << "A pessoa é: " << pessoa.defina_meu_tipo() << std::endl;
    std::cout << "O estudante é: " << estudante.defina_meu_tipo() << std::endl;
    f(pessoa);
    f(estudante);
    return 0;
}
```



# Conversão de tipos

## Downcasting

- Ocorre no sentido Subclasse ⇒ Classe
- Não é feito de forma automática!
- Deve-se deixar explícito, informando o nome do subtipo antes do nome da variável

# Conversão de tipos

## Downcasting

- Nem sempre uma superclasse poderá assumir o tipo de uma subclasse
- Todo Aluno é uma Pessoa
- Nem toda Pessoa é Professor
- Caso não seja possível
  - C++ é esquisito, o erro ocorre no futuro.
  - Use <dynamic\_cast>

# Dynamic Cast

A primeira conversão é ok

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("John Doe", 20180101);

    Pessoa &p2 = estudante;
    std::cout << p2.defina_meu_tipo() << std::endl;

    Estudante &e2 = dynamic_cast<Estudante&>(p2);

    return 0;
}
```

# Dynamic Cast

A segunda é explícita

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);

    Pessoa &p2 = estudante;
    std::cout << p2.defina_meu_titulo() << std::endl;

    Estudante &e2 = dynamic_cast<Estudante&>(p2);

    return 0;
}
```

# Dynamic Cast

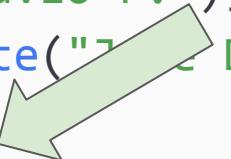
- Mais Seguro em C++
- Você pode fazer um cast <tipo> X
  - Sem usar `dynamic_cast`
  - Chamado de um `static cast`
- Para entender a necessidade é bom entender como funciona polimorfismo

# Exemplo Abaixo

Note que removemos a referência. Qual a saída?!

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);

    Pessoa p2 = estudante;
    std::cout << p2.defina_meu_tipo() << std::endl;
    return 0;
}
```

# Polimorfismo e Late Binding

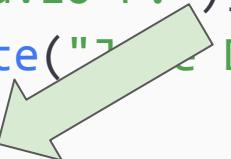
- Polimorfismo funciona bem quando fazemos uso de referências e ponteiros
- Um ponteiro nada mais é do que um endereço um inteiro, então com `dynamic_cast` a linguagem verifica se é possível
- Sem isso, C++ faz a conversão e deixa acontecer

# Exemplo Abaixo

Com ponteiros funciona

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("John Doe", 20180101);

    Pessoa *p2 = &estudante;
    std::cout << p2->defina_meu_tipo() << std::endl;
    return 0;
}
```

# Dynamic Cast

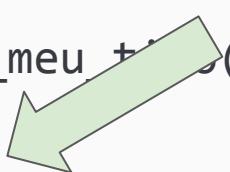
Caso dê errado, e2 é nullptr

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);

    Pessoa *p2 = &estudante;
    std::cout << p2->defina_meu_endereco() << std::endl;


    Estudante *e2 = dynamic_cast<Estudante*>(p2);

    return 0;
}
```

# Static Cast

Caso dê errado, e2 é lixo

```
#include <iostream>

#include "estudante.h"
#include "pessoa.h"

int main() {
    Pessoa pessoa("Flavio F.");
    Estudante estudante("Jane Doe", 20180101);

    Pessoa *p2 = &estudante;
    std::cout << p2->defina_meu_endereco() << std::endl;

    Estudante *e2 = static_cast<Estudante*>(p2);

    return 0;
}
```

# Nas próximas aulas

- Interfaces
- Classes Abstratas
- + Polimorfismo

**Até a prova!**

---