

# Ponteiros e Retornos const

Paulo Ricardo Lisboa de Almeida

# Problema

- Você consegue identificar o problema que criamos com as funções a seguir em *Disciplina.hpp*?

```
//...  
class Disciplina {  
    public:  
        //...  
  
        void adicionarAluno(Pessoa* aluno);  
        void removerAluno(Pessoa* aluno);  
        void removerAluno(unsigned long cpf);  
        std::list<Pessoa*> & getAlunos();  
  
    private:  
        //...  
};  
#endif
```

# Problema


Para economizar memória e processamento, retornamos uma referência para um dado membro.  
Mas através dessa referência, a classe cliente pode fazer o que quiser com esse dado membro que é **privado**!

**Quebramos o encapsulamento!**

```
//...
class Disciplina{
    public:
        //...

        void adicionarAluno(Pessoa* aluno);
        void removerAluno(Pessoa* aluno);
        void removerAluno(unsigned long cpf);
        std::list<Pessoa*> & getAlunos();

    private:
        //...
};
#endif
```



# Teste você mesmo

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

```
#include "Disciplina.hpp"
#include "Pessoa.hpp"
```

```
int main(){
    Disciplina d{"C++"};
    Pessoa p1{"Joao"};

    d.adicionarAluno(&p1);
    std::list<Pessoa*> &alunos{d.getAlunos()};//acessamos a lista original
    Pessoa p2{"Maria"};
    alunos.push_back(&p2);//modificamos o objeto interno de Disciplina

    std::list<Pessoa*>::iterator it = d.getAlunos().begin();
    for( ;it != d.getAlunos().end(); it++){
        std::cout << (*it)->getNome() << std::endl;
    }

    return 0;
}
```

# Indicando que o retorno é const

- Ao inserir *const* **na frente** da declaração da função, indicamos que o **retorno** da função é ***const***
  - Somente leitura
    - Caso o retorno seja um objeto, funções membro não *const* do objeto não podem ser invocadas

# Exemplo

## Disciplina.hpp

```
const std::list<Pessoa*>& getAlunos() const;
```

## Disciplina.cpp

```
const std::list<Pessoa*>& Disciplina::getAlunos() const{  
    return alunos;  
}
```

# Exemplo

## Disciplina.hpp

```
const std::list<Pessoa*>& getAlunos() const;
```

## Disciplina.cpp

```
const std::list<Pessoa*>& Disciplina::getAlunos() const{  
    return alunos;  
}
```

Dois *consts*!!! O que isso significa?

# Exemplo

## Disciplina.hpp

```
const std::list<Pessoa*>& getAlunos() const;
```

## Disciplina.cpp

```
const std::list<Pessoa*>& Disciplina::getAlunos() const{  
    return alunos;  
}
```

O retorno vai ser *const* (somente leitura)

A função é *const* (não altera o estado do objeto quando executada)



# Teste você mesmo

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

```
#include "Disciplina.hpp"
#include "Pessoa.hpp"
```

```
int main(){
    Disciplina d{"C++"};
    Pessoa p1{"Joao"};

    d.adicionarAluno(&p1);
    const std::list<Pessoa*> &alunos{d.getAlunos()};
    //alunos.push_back(&p2); //Isso não é válido

    std::list<Pessoa*>::const_iterator it = alunos.begin();
    for( ;it != alunos.end(); it++){
        std::cout << (*it)->getNome() << std::endl;
    }

    return 0;
}
```

# Teste você mesmo

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

```
#include "Disciplina.hpp"
#include "Pessoa.hpp"
```

```
int main(){
    Disciplina d{"C++"};
    Pessoa p1{"Joao"};
```

Referência const para uma lista

A lista não pode ser alterada

É necessário um `const_iterator` para iterar na lista

```
    d.adicionarAluno(&p1);
    const std::list<Pessoa*> &alunos{d.getAlunos()};
    //alunos.push_back(&p2);//Isso não é válido
```

```
    std::list<Pessoa*>::const_iterator it = alunos.begin();
    for( ;it != alunos.end(); it++){
        std::cout << (*it)->getNome() << std::endl;
    }
```

```
    return 0;
```

```
}
```

# Valores retornados por cópia vs. referência

- Geralmente valores retornados por cópia não são `const`
  - Mas é comum termos retornos de ponteiro ou referência `const`

# Ponteiros const

- Temos quatro cenários possíveis para ponteiros *const*

# Ponteiro não const para dados não const

- Um **ponteiro não *const*** para **dados não const** é o que estamos fazendo até o momento
  - Podemos fazer com que o ponteiro **aponte para qualquer objeto**
  - **Podemos alterar os dados** do objeto apontado
- Exemplo

```
int main(){
    Pessoa p1{"João"};
    Pessoa p2{"Maria"};

    Pessoa* ptr1{&p1};
    ptr1->setNome("Pedro");

    std::cout << ptr1->getNome() << std::endl;
    ptr1 = &p2;
    std::cout << ptr1->getNome() << std::endl;

    return 0;
}
```

# Ponteiro não const para dados const

- Ponteiro não *const* para **dados const**
  - Podemos fazer com que o ponteiro **aponte para qualquer objeto**
  - **Não podemos alterar os dados** do objeto apontado
    - Ex.: Não podemos chamar funções membro não const
  - Declaramos adicionando **const** na frente da declaração do ponteiro

- Exemplo

```
int main(){  
    Pessoa p1{"João"};  
    Pessoa p2{"Maria"};
```

Set não é const, então  
essa chamada não é válida

```
    const Pessoa* ptr1{&p1};  
    //ptr1->setNome("Pedro");
```

O ponteiro pode apontar  
para outros objetos

```
    std::cout << ptr1->getNome() << std::endl;  
    ptr1 = &p2;  
    std::cout << ptr1->getNome() << std::endl;
```

```
    return 0;
```

```
}
```

# Ponteiro **const** para dados não const

- Ponteiro **const** para dados não const
  - O ponteiro não pode trocar de objeto apontado
    - Depois de inicializado, sempre aponta para a mesma região da memória
    - Deve ser inicializado assim que criado
  - **Podemos** alterar os dados do objeto apontado
  - Declaramos adicionando **const** após o tipo da variável

- Exemplo

Podemos alterar os dados  
do objeto apontado

Tentar trocar o objeto  
apontado resultaria em um  
erro de compilação

```
int main(){
    Pessoa p1{"João"};
    Pessoa p2{"Maria"};

    Pessoa* const ptr1{&p1};
    ptr1->setNome("Pedro");

    std::cout << ptr1->getNome() << std::endl;
    //ptr1 = &p2;

    return 0;
}
```

# Finalmente

- O que isso significa?

```
const Pessoa* const ptr1{&p1};
```



# Ponteiro const para dados const

- Ponteiro **const** para dados const
  - O ponteiro não pode trocar de objeto apontado
  - **Não podemos** alterar os dados do objeto apontado
  - Declaramos adicionando **const** antes e após o tipo da variável
- Exemplo

Ambas as linhas  
comentadas resultariam em  
erros de compilação

```
int main(){
    Pessoa p1{"João"};
    Pessoa p2{"Maria"};

    const Pessoa* const ptr1{&p1};
    //ptr1->setNome("Pedro");

    std::cout << ptr1->getNome() << std::endl;
    //ptr1 = &p2;

    return 0;
}
```

# Exemplo

- A função membro retorna um ponteiro const (o objeto não pode ser alterado através do ponteiro) para uma pessoa
  - A função em si também não altera o estado do objeto

```
const Pessoa* getProfessor() const;
```

# Exercício

1. Considere o protótipo da função membro a seguir e explique o objetivo de cada *const* na função

```
const double* calcularImposto(const Investimento* const inv) const;
```

2. Adicione o modificador *const* em todos os trechos do seu projeto onde isso fizer sentido (retornos *const*, funções *const*, parâmetros *const*, ponteiros *const*, ...)

# Referências

- DEITEL, P.; DEITEL, H. **C++ how to Program**. [S.l.]: Pearson, 2017. ISBN 9780134448237
- STROUSTRUP, B. **The C++ Programming Language**. Pearson Education, 2013. ISBN 9780133522853.