


UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Programação Orientada a Objetos (Gerenciando Memória)


Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br



DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

Introdução

- Gerenciamento explícito da memória
 - new / delete (single variables)
 - new[] / delete[] (array variables)
- Prós/Cons
 - Uso eficiente da memória
 - Fácil ter programas problemáticos
- Requer compreensão do modelo de memória e das operações próprias de gerenciamento




PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

2

Introdução

- Mau gerenciamento de memória
 - Usar variáveis não inicializadas
 - Alocar memória e não excluí-la
 - Tentar acessar um valor após excluído
- Boas práticas
 - Sempre inicializar as variáveis
 - Sempre liberar a memória após o uso
 - Certifique-se sempre de que a memória não está mais em uso antes de excluí-lo!




PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

3

Introdução

- Construtores
 - Responsáveis por inicializar os membros do objeto após a alocação na memória
- Construtor de cópia
 - Padrão / User-defined
 - Recebe um objeto como argumento e copia os valores dos membros para o outro objeto
 - Shallow / Deep
 - Copiar endereço / Copiar valor em novo endereço

[https://en.wikipedia.org/wiki/Copy_constructor_\(C%2B%2B\)](https://en.wikipedia.org/wiki/Copy_constructor_(C%2B%2B))



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

4

Introdução


Exemplo 1

```

class ClasseTeste {
public:
    int x, *p;

    ClasseTeste() {
        this->p = new int;
    }

    void display() {
        cout << this->x << " * " << *this->p << endl;
    }
};
  
```



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

5

Introdução

Exemplo 1

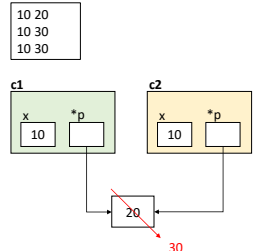
```


int main() {
    ClasseTeste c1;
    c1.x = 10;
    *c1.p = 20;
    c1.display();

    ClasseTeste c2 = c1;
    *c2.p = 30;
    c2.display();

    c1.display();

    return 0;
}
  
```





PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

6

Introdução

Exemplo 1

```
class ClasseTeste {
public:
    int x, *p;

    ClasseTeste() {
        this->p = new int;
    }

    ClasseTeste(ClasseTeste &source) {
        this->x = source.x;
        this->p = new int;
        *this->p = *source.p;
    }

    void display() {
        cout << this->x << " " << *this->p << endl;
    }
};
```

Construtor de cópia

<https://wandbox.org/pemlink/478E3TAXoZERY7g>

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

7

Introdução

■ Regra simples

- Sempre que o operador new for usado, deve-se ser capaz de identificar quando a exclusão será feita

■ Formas de evitar problemas

- Ocultar a alocação de memória em um objeto, fazendo com que o objeto seja responsável
 - Ao ser destruído, ele deve excluir essa memória
- Manter uma contagem das referências
 - Operadores/ferramentas auxiliares da linguagem

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

8

Introdução

■ Destrutores

- Responsáveis por desalocar qualquer memória dinâmica (ponteiros) associada aos membros

■ Utilize, mas entenda os riscos

- Não são chamados em algumas situações
 - Término prematuro do programa (exit)
 - Lançamento de exceção no construtor
 - Remoção por um ponteiro base, sem dtor virtual

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

9

Introdução

Exemplo 2

```
class ClasseTeste {
public:
    int *x, *p;

    ClasseTeste() {
        this->x = new int;
        if (this->x == nullptr) {
            cout << "Memoria insuficiente!" << endl;
            exit(1);
        }

        this->p = new int;
        if (this->p == nullptr) {
            cout << "Memoria insuficiente!" << endl;
            exit(1);
        }
    }

    ~ClasseTeste() {
        delete this->x;
        delete this->p;
    }
};
```

<https://wandbox.org/pemlink/D11A24QmEPMUy8U>

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

10

Introdução

Exemplo 3

```
Circunferencia.hpp
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define CIRCUNFERENCIA_H
#define CIRCUNFERENCIA_H

#include <cmath>
#include "Ponto.hpp"

class Circunferencia {
public:
    Ponto* _centro;
    double _raio;

    Circunferencia(Ponto* centro, double raio);
    ~Circunferencia();

    double calcularArea();
};

#endif

Circunferencia.cpp
#include "Circunferencia.hpp"

Circunferencia::Circunferencia(Ponto* centro, double raio) {
    this->_centro = centro;
    this->_raio = raio;
}

Circunferencia::~Circunferencia() {
    delete this->_centro;
}

double Circunferencia::calcularArea() {
    return M_PI * pow(this->_raio, 2);
}
```

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

11

Introdução

Exemplo 3

```
main.cpp
#include <iostream>
#include "Ponto.hpp"
#include "Circunferencia.hpp"

using namespace std;

int main() {
    Circunferencia* c1 = new Circunferencia(new Ponto(), 10);
    cout << c1->calcularArea() << endl;

    Ponto* p1 = new Ponto(10, 10);
    Circunferencia* c2 = new Circunferencia(p1, 10);
    cout << c2->calcularArea() << endl;

    delete c1;
    delete c2;

    return 0;
}
```

- Ponteiro externo?
- Instanciar dentro de Circunferencia?

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

12

Rule of Three

- Se um (ou mais) dos seguintes membros foi definido, provavelmente deve-se definir todos os três:
 - Destruitor
 - Construtor de cópia
 - Operador de cópia
- Se a versão gerada pelo compilador para uma não se ajusta às necessidades da classe em questão, então provavelmente as outras funções padrões também não

[https://en.wikipedia.org/wiki/Rule_of_three_\(C%2B%2B_programming\)](https://en.wikipedia.org/wiki/Rule_of_three_(C%2B%2B_programming))



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

13

Rule of Three Exemplo

```
class Test {
public:
    Test() {
        cout << "Constructor called." << endl;
    }

    ~Test() {
        cout << "Destructor called." << endl;
    }

    Test(const Test &t) {
        cout << "Copy constructor called." << endl;
    }

    Test& operator = (const Test &t) {
        cout << "Assignment operator called." << endl;
        return *this;
    }
};
```

```
int main() {
    Test t1, t2;
    t2 = t1;
    Test t3 = t1;
    return 0;
}
```

Constructor called.
Constructor called.
Assignment operator called.
Copy constructor called.
Destructor called.
Destructor called.
Destructor called.



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

14

Resource Acquisition Is Initialization (RAII)

- Conceito em C++ que aproveita a propriedade essencial da Pilha para simplificar o gerenciamento de memória de objetos no Heap
- Vincula recursos ao *lifetime* do objeto, que pode não coincidir com a entrada/saída de um escopo
- Princípio
 - Envolver o recurso (um ponteiro) em um outro objeto e descartar o recurso em seu destrutor

<https://en.cppreference.com/w/cpp/language/raii>
https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

15

Smart Pointers (C++11)

- Classes auxiliares que envolvem ponteiros e sobrecarregam os operadores \rightarrow e $*$
- Gerenciamento automático de memória
 - Quando um ponteiro inteligente não está mais em uso (sai do escopo), a memória para a qual ele aponta é desalocada automaticamente
- Tipos
 - `std::unique_ptr`
 - `std::shared_ptr`

https://en.wikipedia.org/wiki/Smart_pointer



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

16

Smart Pointers (C++11)

- `std::unique_ptr`
 - Possui um recurso alocado dinamicamente
 - Apenas ele pode apontar para o recurso
- `std::shared_ptr`
 - Possui um recurso alocado compartilhado
 - Mantém um contador interno com o número de `shared_ptr` que possuem o mesmo recurso



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

17

Smart Pointers (C++11) Exemplo 1

```
class ClasseA {
public:
    int id;

    ClasseA(int id) : id(id) {
        cout << "ClasseA::Constructor:" << this->id << endl;
    }

    ~ClasseA() {
        cout << "ClasseA::Destructor:" << this->id << endl;
    }
};
```



PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

18

Smart Pointers (C++11)

Exemplo 1

```
int main() {
    ClasseA c1(1);

    ClasseA *c2 = new ClasseA(2);

    return 0;
}
```

<https://wandbox.org/permlink/9tE1vcNtLNSaGZ>

```
ClasseA::Constructor:1
ClasseA::Constructor:2
ClasseA::Constructor:3
ClasseA::Destructor:3
ClasseA::Destructor:1
```

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

19

Smart Pointers (C++11)

Exemplo 2

```
int main() {
    unique_ptr<ClasseA> c1(new ClasseA(1));
    // Compile Error: unique_ptr object is not copyable
    //unique_ptr<ClasseA> c2 = c1;

    shared_ptr<ClasseA> c2(new ClasseA(2));
    shared_ptr<ClasseA> c3 = c2;

    cout << c2.use_count() << endl;

    c3 = nullptr;
    cout << c2.use_count() << endl;

    return 0;
}
```

<https://wandbox.org/permlink/vnBJCmmbWgYH7Zh>

```
ClasseA::Constructor:1
ClasseA::Constructor:2
2
1
ClasseA::Destructor:2
ClasseA::Destructor:1
```

DCC

PDS 2 - Programação Orientada a Objetos (Gerenciando Memória)

20

Smart Pointers (C++11)

Exemplo 3

```
class Animal {
public:
    virtual void fale() {
        cout << "Fale padrao!" << endl;
    };

    ~Animal() {
        cout << "Animal::Destructor" << endl;
    };
};
```

```
class Gato : public Animal {
public:
    void fale() override {
        cout << "Miau!" << endl;
    };

    ~Gato() {
        cout << "Gato::Destructor" << endl;
    };
};
```

```
class Cachorro : public Animal {
public:
    void fale() override {
        cout << "Au! Au!" << endl;
    };

    ~Cachorro() {
        cout << "Cachorro::Destructor" << endl;
    };
};
```

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 1/2)

21

Smart Pointers (C++11)

Exemplo 3

```
#include <list>

int main() {
    list<Animal*> lista;

    for(int i=0; i<5;i++) {
        if (i % 2 == 0)
            lista.push_back(new Cachorro());
        else
            lista.push_back(new Gato());
    }

    for (auto a : lista)
        a->fale();

    return 0;
}
```

```
Au! Au!
Miau!
Au! Au!
Miau!
Au! Au!
```

Memory Leak!
Nenhum destrutor é chamado!

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 1/2)

22

Smart Pointers (C++11)

Exemplo 3

```
int main() {
    list<unique_ptr<Animal>> lista;

    for(int i=0; i<5;i++) {
        if (i % 2 == 0)
            lista.push_back(unique_ptr<Animal>(new Cachorro()));
        else
            lista.push_back(unique_ptr<Animal>(new Gato()));
    }

    for (auto const &a : lista)
        a->fale();

    return 0;
}
```

```
Au! Au!
Miau!
Au! Au!
Miau!
Au! Au!
Animal::Destructor
Animal::Destructor
Animal::Destructor
Animal::Destructor
```

Memory Leak!
Destrutor derivado não é chamado!

DCC

PDS 2 - Programação Orientada a Objetos (Polimorfismo - 1/2)

23

Smart Pointers (C++11)

Exemplo 3

```
class Animal {
public:
    virtual void fale() {
        cout << "Fale padrao!" << endl;
    };

    virtual ~Animal() {
        cout << "Animal::Destructor" << endl;
    };
};
```

<https://wandbox.org/permlink/FHSEIAZigqY8aKX>

```
Au! Au!
Miau!
Au! Au!
Miau!
Au! Au!
Cachorro::Destructor
Animal::Destructor
Gato::Destructor
Animal::Destructor
Cachorro::Destructor
Animal::Destructor
Gato::Destructor
Animal::Destructor
Cachorro::Destructor
Animal::Destructor
```

DCC

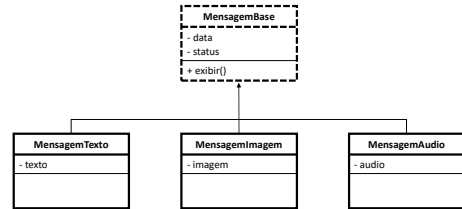
PDS 2 - Programação Orientada a Objetos (Polimorfismo - 1/2)

24

Exercício



Exercício



Exercício

```

MensagemBase.hpp
#ifndef MENSAGEMBASE_H
#define MENSAGEMBASE_H
#include <ctime>

enum MsgStatus {
    NONE,
    SENT,
    DELIVERED,
    READ
};

class MensagemBase {
private:
    std::time_t _data;
    MsgStatus _status;

public:
    virtual void exibir() = 0;

    MensagemBase();
    std::time_t getData();
    MsgStatus getStatus();
};

#endif
  
```

Exercício

```

MensagemBase.cpp
#include "MensagemBase.hpp"

MensagemBase::MensagemBase() {
    this->_data = std::time(nullptr);
    this->_status = MsgStatus::NONE;
}

std::time_t MensagemBase::getData() {
    return this->_data;
}

MsgStatus MensagemBase::getStatus() {
    return this->_status;
}
  
```

Exercício

```

Chat.hpp
#ifndef CHAT_H
#define CHAT_H

#include <iostream>
#include "MensagemBase.hpp"
#include "MensagemTexto.hpp"
#include "MensagemImagem.hpp"
#include "MensagemAudio.hpp"

class Chat {
private:
    std::list<MensagemBase*> _historico;
    void enviaMensagem(MensagemBase* msg);

public:
    void exibirMensagens();

    void enviaMensagemTexto(std::string texto);
    void enviaMensagemImagem(std::string imagem);
    void enviaMensagemAudio(std::string audio);
};

#endif
  
```

Exercício

```

Chat.cpp
#include "Chat.hpp"

void Chat::enviaMensagem(MensagemBase* msg) {
    this->_historico.push_back(msg);
}

void Chat::exibirMensagens() {
    for (auto m : this->_historico) {
        time_t d = m->getData();
        char data[20];
        strftime(data, 20, "%Y-%m-%d %H:%M:%S", localtime(&d));

        std::cout << "!" << data << ", " << m->getStatus() << "!" << std::endl;
        m->exibir();
    }
}

void Chat::enviaMensagemTexto(std::string texto) {
    MensagemTexto* msg = new MensagemTexto(texto);
    this->enviaMensagem(msg);
}

void Chat::enviaMensagemImagem(std::string imagem) {
    MensagemImagem* msg = new MensagemImagem(imagem);
    this->enviaMensagem(msg);
}

void Chat::enviaMensagemAudio(std::string audio) {
    MensagemAudio* msg = new MensagemAudio(audio);
    this->enviaMensagem(msg);
}
  
```

Exercício

```
main.cpp
#include "Chat.hpp"

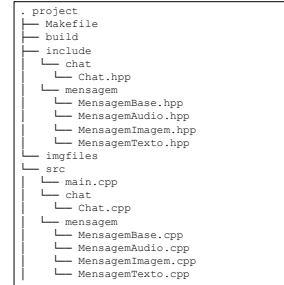
int main() {
    Chat chat;

    chat.enviarMensagemTexto("Oi, tem aula de PDS2 hoje?");
    chat.enviarMensagemAudio("audio.wav");
    chat.enviarMensagemImagem("imagem03.ascii");
    chat.enviarMensagemTexto("Mas que puta :(");

    chat.exibirMensagens();

    return 0;
}
```

Exercício



Exercício

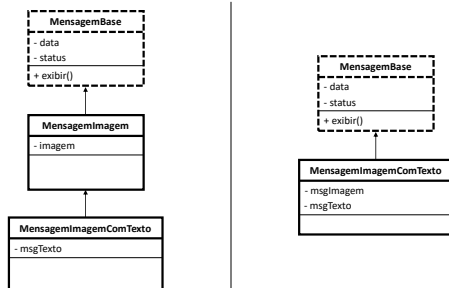
```

1 #include <string>
2 #include <ctime>
3
4 class MensagemBase {
5 private:
6     std::time_t _data;
7     std::string _avatar;
8 public:
9     MensagemBase(std::string avatar);
10    std::time_t get_data() const;
11    virtual std::string get_avatar() const;
12    virtual void exibir() const = 0;
13 };
14
15 #endif
16
17 #pride
18
19 23:15 ✓
```

Exercício



Exercício



Exercício

- Tarefas
 - Faça as alterações necessárias no Makefile
 - Modifique Chat
 - Enviar MensagemImagemComTexto
 - Alterar o Status das mensagens
 - Como especializar Chat?
 - Conversa Individual / Grupo
 - Verifique (corrija) a presença de memory leaks
 - Modifique para utilizar smart_pointers