# Construtores

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

# Inicializar valores

- Como em C, variáveis e objetos não são inicializados automaticamente
  - Possuem lixo de memória
- Para o caso de objetos, temos 3 opções para inicializar os dados membro
  1. Não inicializar e deixar a cargo do programador dar os valores iniciais através dos gets e sets, por exemplo
     - Problemas?

# Inicializar valores

- Como em C, variáveis e objetos não são inicializados automaticamente
  - Possuem lixo de memória
- Temos algumas opções para inicializar os dados membro
  1. Não inicializar e deixar a cargo do programador dar os valores iniciais através dos gets e sets, por exemplo
     - Problemas?

# Inicializar valores

- Como em C, variáveis e objetos não são inicializados automaticamente
  - Possuem lixo de memória
- Temos algumas opções para inicializar os dados membro
  1. Não inicializar e deixar a cargo do programador dar os valores iniciais através dos gets e sets, por exemplo
     - Se o programador esquecer de inicializar, vai operar com lixo

# Inicializar valores

- Como em C, variáveis e objetos não são inicializados automaticamente
  - Possuem lixo de memória
- Temos algumas opções para inicializar os dados membro
  1. Não inicializar e deixar a cargo do programador dar os valores iniciais através dos gets e sets, por exemplo
     - Se o programador esquecer de inicializar, vai operar com lixo
  2. Dar valores iniciais através de construtores
     - Discutiremos na aula de hoje
  3. Forçar o usuário a passar dados iniciais através do construtor
     - Discutiremos na aula de hoje

# Construtor

- Um construtor é uma ***função membro*** especial
  - É **chamada automaticamente** quando um objeto é instanciado
    - Quando um objeto é instanciado, a memória para o objeto é alocada, e o construtor é chamado
      - **Automaticamente e nessa ordem**

# Construtor

- Um construtor deve ter **o mesmo nome da classe**
- Um construtor **não possui tipo de retorno** (nem void)
- Construtores comumente são públicos
  - Veremos no decorrer da disciplina alguns cenários onde faz sentido um construtor privado

# Pessoa .hpp e .cpp

Pessoa.hpp

```
#ifndef PESSOA_H
#define PESSOA_H

class Pessoa{
    public:
        Pessoa();//construtor

        unsigned char getIdade();
        void setIdade(unsigned char novaIdade);

        //...
};

#endif
```

Pessoa.cpp

```
#include "Pessoa.hpp"

Pessoa::Pessoa(){
}

std::string Pessoa::getNome(){
    return nome;
}

//...
```

UDESC

# Construtor default

- Foi definido um construtor que não recebe parâmetros e não faz coisa alguma
  - Esse é chamado de construtor ***default*** (padrão)
  - Quando você não cria seus próprios construtores, o compilador **automaticamente injeta um construtor *default*** em sua classe

```
Pessoa::Pessoa(){
}
```

# Faça o teste

- No corpo do construtor de Pessoa, adicione um *cout*
- Instancie um novo objeto do tipo pessoa no main e veja o que acontece
- 3 minutos!

# Adicionando Parâmetros

- Vamos solicitar o nome e a idade da pessoa quando um objeto for instanciado

Em pessoa.hpp

```
Pessoa(std::string nomePessoa, unsigned short int idadePessoa);
```

Em pessoa.cpp

```
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa){
	nome = nomePessoa;
	idade = idadePessoa;
}
```

# Adicionando Parâmetros

- Instancie um objeto do tipo pessoa no main
  - Compile
  - O que acontece?

# Adicionando Parâmetros

- Instancie um objeto do tipo pessoa no main
  - Compile
  - O que acontece?
    - Impossível compilar, pois não podemos construir um objeto do tipo pessoa sem passar seus parâmetros ao construtor

# Passagem de parâmetros para o construtor

- Para passar os parâmetros solicitados, basta adicioná-los entre chaves depois do nome do objeto
- Exemplo
  - Pessoa p1{"Joao", 20};
- Atenção
  - É **convenção** chamar o construtor **no C++11** utilizando as chaves
    - Antes do C++11, utilizava-se parêntesis – que ainda são válidos
      - Ao utilizar um compilador que está com a versão do C++11 desabilitada, utilize ( )
  - Se o construtor de lista estiver habilitado, ele será automaticamente invocado
    - Veremos nos próximos slides

# Indo mais rápido

- Os parâmetros passados para o construtor podem ser utilizados na lista de inicializador de membro (member-initializer list)
  - Melhor do que inicializar dentro do corpo do construtor como fizemos
    - Inicializar os membros nessa lista é mais eficiente
    - Alguns membros de dados só podem ser inicializados nessa lista
    - A lista é **executada antes** do corpo do construtor

# Member-initializer list

- Para usar a lista de inicializador de membro no construtor
  - No .cpp

```
NomeClasse::NomeConstrutor(tipo par1, tipo par2, …)
        :nomeMembro1{par1}, nomeMembro2{par2}, … {
        //corpo do construtor aqui
}
```

Lista de inicializador de membro

# Member-initializer list

- A lista de inicializador é automaticamente chamada quando você fizer uma chamada ao construtor utilizando chaves { … }
  - Exemplo
    Pessoa p1{"Joao", 20};
- Quando usamos a lista de inicializador de membro, o compilador pode alinhar as variáveis na memória
  - E pode operar com todos os dados de uma vez quando inicializar a classe, **se ele achar que isso é mais eficiente**

# Member-initializer list

- Exemplo para a classe Pessoa
  - Os dados membro devem aparecer na lista **na mesma ordem** em que são declarados no header
    - Caso contrário o compilador vai acusar um *warning* informando que você quer inicializar em uma ordem, mas na verdade uma ordem diferente vai ser utilizada
      - O compilador na verdade vai trocar a ordem dos parâmetros para você
      - O warning pode ser ignorado, mas vai ficar poluindo o seu log

```
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa)
    :nome{nomePessoa}, cpf{0}, idade{(unsigned char)idadePessoa} {
}
```

# Adicionando cpf

- Adicione o cpf como parâmetro do construtor
  - Quais os possíveis problemas?

# Adicionando cpf

- Adicione o cpf como parâmetro do construtor
  - Quais os possíveis problemas?
    - Como vamos validar o cpf?
    - Podemos validar, mas não podemos retornar um booleano indicando se tudo deu certo ou não
      - O construtor não possui retorno
    - A solução para o problema está no lançamento de exceções
      - Veremos no futuro
    - No momento, simplesmente inicialize com 0 o cpf caso o parâmetro seja um cpf inválido
- 3 minutos!

# Adicionando CPF

Em Pessoa.hpp

```
Pessoa(std::string nomePessoa,
       unsigned short int idadePessoa,
       unsigned long cpfPessoa);
```

Em Pessoa.cpp

```
Pessoa:: Pessoa(std::string nomePessoa,
                unsigned short int idadePessoa, unsigned long cpfPessoa)
                :nome{nomePessoa}, idade{(unsigned char)idadePessoa} {
        if(validarCPF(cpfPessoa))
                cpf = cpfPessoa;
        else
                cpf = 0;
}
```

O cpf exige lógica para validação, então ele não pode ser inicializador na lista de inicializador de membro

UDESC

# Construtores

- Se você definir um construtor, **o construtor padrão não será injetado** pelo compilador
  - Mas podemos ter mais de um construtor
    - Conceito de **sobrecarga de funções**
      - Veremos a sobrecarga em detalhes adiante, mas por enquanto vamos aplicar o básico nos construtores

# Múltiplos construtores

- Os múltiplos construtores devem
  - Ter diferentes tipos e/ou números de parâmetros
  - Ter o mesmo nome

# Múltiplos construtores - exemplo

Em Pessoa.hpp

```
Pessoa();
Pessoa(std::string nomePessoa);
Pessoa(std::string nomePessoa, unsigned short int idadePessoa);
```

Em Pessoa.cpp

```
Pessoa::Pessoa(){
}
Pessoa::Pessoa(std::string nomePessoa)
	:nome{nomePessoa} {
}
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa)
	:nome{nomePessoa}, cpf{0}, idade{(unsigned char)idadePessoa} {
}
```

# No main

```
int main(){
    Pessoa p1{"Joao", 20};
    Pessoa p2{"Maria"};
    Pessoa p3;

    std::cout << "P1: " << p1.getNome() << " " << p1.getIdade() << std::endl;
    std::cout << "P2: " << p2.getNome() << " " << p2.getIdade() << std::endl;
    std::cout << "P3: " << p3.getNome() << " " << p3.getIdade() << std::endl;

    return 0;
}
```

# Teste

- Compile e execute
  - Como o compilador sabe qual construtor chamar?

# Teste

- Compile e execute
  - Como o compilador sabe qual construtor chamar?
    - Pela quantidade e tipo dos parâmetros passados ao construir os objetos

# Questão

- Qual o problema com os construtores a seguir (verifique sem tentar compilar)
  - Obs.: assuma que por algum motivo desejamos ter um construtor onde a idade é passada como long

```
Pessoa();

Pessoa(unsigned long cpf);

Pessoa(unsigned long IdadeEmLong);

Pessoa(std::string nomePessoa);
```

# Questão

- Construtores devem ter o número de parâmetros e/ou tipos dos parâmetros diferentes
  - Os dois construtores assinalados possuem 1 parâmetro do tipo unsigned long
    - O nome dos parâmetros é irrelevante (lembre-se que para a máquina nomes de parâmetros e variáveis não existem).
      - Aprendam mais sobre isso na disciplina de arquitetura e organização de computadores e de Microprocessadores
        - Dizem que o professor é muito bom ;)

```
Pessoa();

Pessoa(unsigned long cpf);

Pessoa(unsigned long IdadeEmLong);

Pessoa(std::string nomePessoa);
```

# Inicializar valores

- Agora que você aprendeu sobre construtores, responda:
- Temos algumas opções para inicializar os dados membro
  1. Não inicializar e deixar a cargo do programador dar os valores iniciais através dos gets e sets, por exemplo
     - Se o programador esquecer de inicializar, vai operar com lixo
  2. Dar valores iniciais através de construtores
     - **Quais as vantagens e desvantagens?**
  3. Forçar o usuário a passar dados iniciais através do construtor
     - **Quais as vantagens e desvantagens?**

# Inicializar valores

- Agora que você aprendeu sobre construtores, responta:
- Temos algumas opções para inicializar os dados membro
  1. Não inicializar e deixar a cargo do programador dar os valores iniciais através dos gets e sets, por exemplo
     - Se o programador esquecer de inicializar, vai operar com lixo
  2. Dar valores iniciais através de construtores
     - E se os valores iniciais dados não são úteis para o programador?
       - Gastamos processamento a toa, e o programador ainda vai precisar fazer gets e sets
  3. Forçar o usuário a passar dados iniciais através do construtor
     - O programador vai precisar passar valores válidos para a o objeto
       - Isso nem sempre é bom
         - As vezes o programador só quer um “objeto padrão” para realizar alguma tarefa

# Delegating Constructor

- Delegating Constructor (construtor delegado)
  - Disponível a partir do C++11
  - Um **construtor pode chamar outro construtor da mesma classe**
  - Especialmente útil por que um construtor “mais completo” (que recebe mais parâmetros por exemplo) pode estar fazendo o mesmo trabalho que outro construtor, apenas com um extra
  - Usar construtores delegados é uma boa prática do ponto de vista da engenharia de software
    - Evita duplicidade de código e facilita a manutenção
- Para usar
  - Chame o construtor usando : depois do fecha parêntesis dos parâmetros do construtor

# Exemplo

## Sem construtor delegado

```
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa)
    :nome{nomePessoa}, idade{(unsigned char)idadePessoa} {
}
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa, unsigned long cpfPessoa)
        : nome{nomePessoa}, idade{(unsigned char)idadePessoa} {
    if(validarCPF(cpfPessoa))
        cpf = cpfPessoa;
    else
        cpf = 0;
}
```

Código Replicado!

## Com construtor delegado

```
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa)
    :nome{nomePessoa}, idade{(unsigned char)idadePessoa} {
}
Pessoa:: Pessoa(std::string nomePessoa, unsigned short int idadePessoa, unsigned long cpfPessoa)
        : Pessoa(nomePessoa, idadePessoa){
    if(validarCPF(cpfPessoa))
        cpf = cpfPessoa;
    else
        cpf = 0;
}
```

UDESC

# Teste você mesmo

- Adicione couts nos construtores e veja a ordem em que os construtores são chamados

# Para a próxima aula

1. Coloque couts nos seus construtores de Pessoa, e crie objetos no *main*. Veja quais construtores são chamados dependendo dos objetos que você criou.

2. Modifique a classe retângulo solicitada em aulas passadas para que ela possua um construtor *default* (que não recebe parâmetros) e um construtor que recebe os dados do retângulo.
   - Inicialize os dados do retângulo no construtor *default* também, utilizando algum valor válido fixo (exemplo: largura e altura = 0)

3. Pesquise sobre construtores com **argumentos** (parâmetros) default
   - Exemplo

     *Pessoa(std::string nome, usigned short int idade = 18)*
     - Modifique a classe Retângulo para que pelo menos um dos argumentos de um de seus construtores possua um valor *default*

# 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.