Construtores

Inicializar valores

Como em C, variáveis e objetos não são inicializados automaticamente, exceto em alguns casos específicos.

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, exceto em alguns casos específicos.

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: o programador pode esquecer de inicializar.

Inicializar valores

Como em C, variáveis e objetos não são inicializados automaticamente, exceto em alguns casos específicos.

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: o programador pode esquecer de inicializar.
2. Dar valores iniciais através de construtores.
   Aula de hoje.
3. Forçar o usuário a passar dados iniciais através do construtor.
   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 geralmente são públicos.

Veremos no decorrer da disciplina alguns cenários onde faz sentido um construtor não público.

Exemplo

```
#ifndef PESSOA_H
#define PESSOA_H

#include<string>

class Pessoa{
   public:
       Pessoa();

       unsigned long getCpf();
       void setCpf(unsigned long novoCpf);

       std::string getNome();
       void setNome(std::string novoNome);

       unsigned short int getIdade();
       void setIdade(unsigned short int novaIdade);
   private:
       bool validarCPF(unsigned long cpfTeste);

       std::string nome;
       unsigned long cpf;
       unsigned char idade;
};
#endif
```

```
#include "Pessoa.hpp"

#include <iostream>

Pessoa::Pessoa(){
}

unsigned long Pessoa::getCpf(){
   //retorna uma cópia do cpf
   return cpf;
}

void Pessoa::setCpf(unsigned long novoCpf){
   if(validarCPF(novoCpf)){
       cpf = novoCpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}

//...
```

Construtor Default

Foi definido um construtor que não recebe parâmetros e não faz coisa alguma.

Construtor **default** (padrão).

Se você não definir construtores, o compilador automaticamente injeta um construtor default em sua classe.

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

Faça você mesmo

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!

Construtor com parâmetros

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

Construtor com parâmetros

```
#ifndef PESSOA_H
#define PESSOA_H

#include<string>

class Pessoa{
   public:
       Pessoa(std::string nome, unsigned short int idade);


       unsigned long getCpf();
       void setCpf(unsigned long novoCpf);

       std::string getNome();
       void setNome(std::string novoNome);

       unsigned short int getIdade();
       void setIdade(unsigned short int novaIdade);
   private:
       bool validarCPF(unsigned long cpfTeste);

       std::string nome;
       unsigned long cpf;
       unsigned char idade;
};
#endif
```

```
#include "Pessoa.hpp"

#include <iostream>

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


unsigned long Pessoa::getCpf(){
   //retorna uma cópia do cpf
   return cpf;
}

void Pessoa::setCpf(unsigned long novoCpf){
   if(validarCPF(novoCpf)){
       cpf = novoCpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}

//...
```

Indo mais rápido

Lista de inicializador de membros (*member-initializer list*).

Melhor do que inicializar dentro do corpo do construtor.

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 .cpp faça:

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

Member-initializer list

A lista de inicializador é **automaticamente chamada** quando você fizer uma chamada ao construtor utilizando chaves.

Exemplo: Pessoa p1{"Joao", 20};

Sempre coloque os parâmetros na lista **na mesma ordem que eles foram declarados na classe**.

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 achar que isso é mais eficiente.

A lista de inicializador de membro evita ainda a chamada de outros construtores desnecessariamente.

Veja na vídeo-aula https://youtu.be/AJ2X2I-mq2A

Construtor com parâmetros

nome pode ser inicializado na member-initializer list, pois é só copiar o dado do parâmetro para o dado membro.

```
#include "Pessoa.hpp"

#include <iostream>

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


unsigned long Pessoa::getCpf(){
   //retorna uma cópia do cpf
   return cpf;
}

void Pessoa::setCpf(unsigned long novoCpf){
   if(validarCPF(novoCpf)){
       cpf = novoCpf;
       return;
   }
   cpf = 0;//indica que não é um cpf válido
   return;
}

//...
```

Instanciando

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 e superiores utilizando as chaves.

Antes do C++11, utilizava-se parêntesis, que ainda são válidos. **Mas não faça isso!**

Construtor com parâmetros

`idade` precisa ser inicializado no corpo da função, já que possui lógica envolvida.

```
#include "Pessoa.hpp"

#include <iostream>

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


unsigned long Pessoa::getCpf(){
    //retorna uma cópia do cpf
    return cpf;
}

void Pessoa::setCpf(unsigned long novoCpf){
    if(validarCPF(novoCpf)){
        cpf = novoCpf;
        return;
    }
    cpf = 0;//indica que não é um cpf válido
    return;
}

//...
```

Faça você mesmo

Adicione o cpf como parâmetro do construtor.

Modifique o main para construir uma Pessoa considerando agora esse novo construtor.

5 minutos!

Múltiplos Construtores

Quando um construtor é definido, **o construtor padrão não será injetado pelo compilador**.

Mas podemos ter mais de um construtor.

Conceito de **sobrecarga de funções.**

Tipo de **polimorfismo**.

Veremos detalhes no futuro, 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

```
class Pessoa{
   public:
       Pessoa(std::string nomePessoa);
       Pessoa(std::string nomePessoa,
            unsigned long cpfPessoa,
            unsigned short int idade);

       //...
   private:
       bool validarCPF(unsigned long cpfTeste);

       std::string nome;
       unsigned long cpf;
       unsigned char idade;
};
```

```
#include "Pessoa.hpp"

#include <iostream>

Pessoa::Pessoa(std::string nomePessoa)
   :nome{nomePessoa}, cpf{0}, idade{0}{
}

Pessoa::Pessoa(std::string nomePessoa,
   unsigned long cpfPessoa,
   unsigned short int idadePessoa)
       :nome{nomePessoa}{
   setCpf(cpfPessoa);
   setIdade(idadePessoa);
}

//...
```

No main

```cpp
#include<iostream>

#include<string>
#include"Pessoa.hpp"

int main(){
   Pessoa p1{"Joao", 11111111111, 20};
   Pessoa p2{"Maria"};

   std::cout << p1.getNome() << '\t' << p1.getIdade() << '\t' <<  p1.getCpf() << std::endl;
   std::cout << p2.getNome() << std::endl;
   return 0;
}
```

Faça você mesmo

Compile e execute.

Como o compilador sabe qual construtor chamar?

Faça você mesmo

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).

```
Pessoa();

Pessoa(unsigned long cpf);

Pessoa(std::string nomeMae);

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

Questão

Construtores devem ter o número de parâmetros e/ou tipos dos parâmetros diferentes.

O **nome** dos parâmetros é **irrelevante**.

Para a máquina nomes de parâmetros e variáveis não existem (tudo vira endereço de memória).

```
Pessoa();
Pessoa(unsigned long cpf);
Pessoa(std::string nomeMae);
Pessoa(std::string nomePessoa);
```

Voltando ao início

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?

Voltando ao início

2. Dar valores iniciais através de construtores.
    E se os valores iniciais dados não são úteis para o programador?
    Gastamos processamento desnecessariamente, e o programador ainda vai precisar fazer gets e sets.

3. Forçar o programador a passar dados iniciais através do construtor.
    O programador vai precisar passar valores válidos para o objeto.
    Nem sempre é bom ou possível.
        Às 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 quando construtores mais complexos (com mais parâmetros por exemplo) apenas adicionam trabalho a construtores mais simples.

Usar construtores delegados é uma **boa prática**.

Evitar duplicidade de código e facilitar a manutenção.

Para usar chame o construtor usando : depois do fecha parênteses dos parâmetros do construtor.

Delegating Constructor

```cpp
#include "Pessoa.hpp"

#include <iostream>

Pessoa::Pessoa(std::string nomePessoa)
   :nome{nomePessoa}{
}

Pessoa::Pessoa(std::string nomePessoa, unsigned long cpfPessoa, unsigned short int idadePessoa)
       :Pessoa{nomePessoa}{
   setCpf(cpfPessoa);
   setIdade(idadePessoa);
}
//...
```

Dica

Você não pode chamar um delegating constructor e ao mesmo tempo inicializar dados membros usando a *member-initializer list*.

Ou você faz um, ou outro.

"A mem-initializer-list can delegate to another constructor of the constructor's class using any class-or-decltype that denotes the constructor's class itself. If a mem-initializer-id designates the constructor's class, it shall be the only mem-initializer; ..."

ISO/IEC JTC1 SC22 WG21 N4860: https://isocpp.org/files/papers/N4860.pdf

Faça você mesmo

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

Exercícios

1. 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.
2. Inicialize os dados do retângulo no construtor default, utilizando algum valor válido fixo (exemplo: largura e altura = 0).
3. Pesquise sobre construtores com argumentos (parâmetros) default.

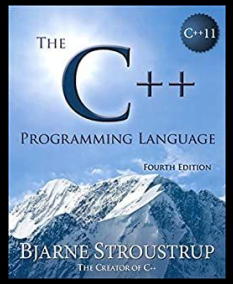
   Exemplo Pessoa(std::string nome, usigned short int idade = 18).

   a. Modifique a classe Retângulo para que pelo menos um dos argumentos de um de seus construtores possua um valor default.
4. Leia sobre como representar construtores no diagrama de classes.
   a. Faça o diagrama de classes UML para a classe retângulo atual.
5. Assista a aula disponibilizada sobre o overhead causado por não se utilizar a lista de inicialização de membro:

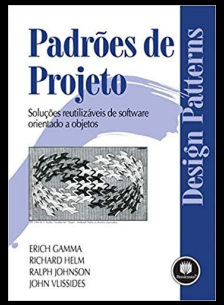
   https://youtu.be/AJ2X2I-mq2A

Referências

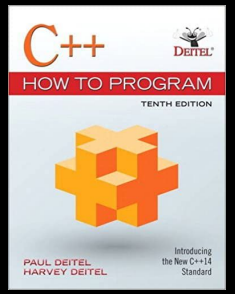
Bjarne Stroustrup. The C++ Programming Language. Addison-Wesley, 2013.



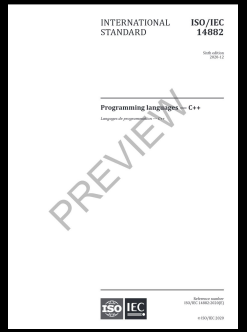
Gamma, E. Padrões de Projetos: Soluções Reutilizáveis. Bookman. 2009.



Deitel, H. M., Deitel, P. J. C++: como programar. 5a ed. Pearson Prentice Hall. 2006.



ISO/IEC 14882:2020 Programming languages - C++: www.iso.org/obp/ui/#iso:std:iso-iec:14882:ed-6:v1:en



Licença