Dados Membro Estáticos

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Conteúdo Ministrado

- Analise a classe conteúdo ministrado disponibilizada com o projeto
  - Representa um conteúdo (ex.: aula sobre ponteiros) que foi ministrado em determinada disciplina, e a quantidade de horas que foram gastas nesse conteúdo

Criando um identificador único

- Vamos adicionar um identificador **único** para cada objeto do tipo *ConteudoMinistrado* criado
- **Desejamos fazer isso de maneira automática**

Dados membro estáticos

- Dados membro estáticos **pertencem a classe**, e não aos objetos
  - Em outras palavras, um dado membro estático é **compartilhado entre todos os objetos da classe** (entre todas as instâncias)
- Para declarar um dado membro como estático, basta adicionar o modificador ***static*** na sua declaração
  - *static tipoDado nomeDado;*

Exemplo

- Adicione o seguinte na classe ConteudoMinistrado
  - Note que o dado está **publico para facilitar os testes**

ConteudoMinistrado.hpp

```
//...
class ConteudoMinistrado{
        public:
                static unsigned int proxId;

                ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo);
                //...
        private:
                std::string descricao;
                unsigned short cargaHorariaConteudo;
                unsigned int id;
};
```

Inicialização

- Dados estáticos são inicializados no cpp
- Não incluímos o modificador static no inicializador
  - *tipoDado NomeClasse::nomeDado{valorInicial}*

Exemplo

ConteudoMinistrado.cpp

```
#include "ConteudoMinistrado.hpp"

unsigned int ConteudoMinistrado::proxId{0};

ConteudoMinistrado::ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo):
        descricao(descricao), cargaHorariaConteudo(cargaHorariaConteudo), id{0}{
}

std::string& ConteudoMinistrado::getDescricao(){
        return descricao;
}

//...
```

UDESC

Acessando os dados estáticos

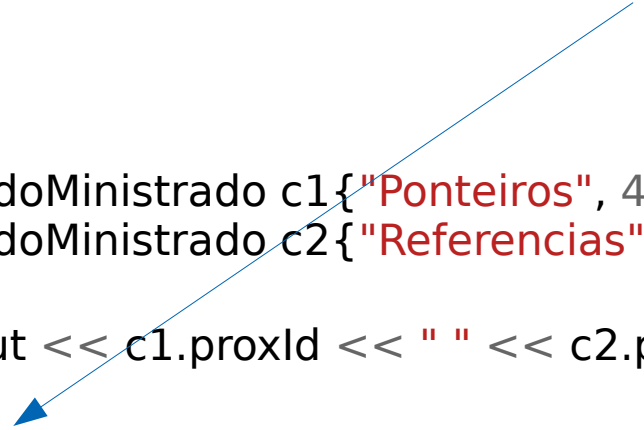
- Os dados (e funções estáticas) podem ser acessados como qualquer dado normal
  - Podemos utilizar, por exemplo, o operador . (ponto)

Teste Você Mesmo

```
//...

int main(){
	ConteudoMinistrado c1{"Ponteiros", 4};
	ConteudoMinistrado c2{"Referencias", 2};

	std::cout << c1.proxId << " " << c2.proxId << std::endl;

	c1.proxId++;
	std::cout << c1.proxId << " " << c2.proxId << std::endl;

	return 0;
}
```

Teste Você Mesmo

Note que alteramos proxId através de c1, mas o dado acessado por c2 também é alterado. Na verdade é o mesmo dado, pois o dado estático pertence a classe, e então existe **apenas uma cópia** desse dado na memória.



```
//...

int main(){
	ConteudoMinistrado c1{"Ponteiros", 4};
	ConteudoMinistrado c2{"Referencias", 2};

	std::cout << c1.proxId << " " << c2.proxId << std::endl;

	c1.proxId++;
	std::cout << c1.proxId << " " << c2.proxId << std::endl;

	return 0;
}
```

Acessando via a Classe

- Acessamos o dado estático via objetos
  - Esse tipo de acesso é **desencorajado**
  - Funciona sem problemas, mas é confuso
    - Passa a impressão que estamos acessando um dado/função membro convencional
- Uma abordagem melhor é realizar acessos através da classe
  - NomeClasse::nomeDadoEstatico;

Exemplo

```
//...
int main(){
	Pessoa prof1{"João", 40};
	ConteudoMinistrado c1{"Ponteiros", 4};
	ConteudoMinistrado c2{"Referencias", 2};

	ConteudoMinistrado::proxId++;//essa é a forma recomendada de acessar os membros estáticos
	std::cout << ConteudoMinistrado::proxId << std::endl;



	return 0;
}
```

Podemos fazer isso?

```
//...
int main(){
	ConteudoMinistrado::proxId++;
	std::cout << ConteudoMinistrado::proxId << std::endl;

	return 0;
}
```

Podemos fazer isso? Note que não temos objetos do tipo *ConteudoMinistrado* na memória!!!

Podemos acessar mesmo sem objetos

```
//...
int main(){
	ConteudoMinistrado::proxId++;
	std::cout << ConteudoMinistrado::proxId << std::endl;

	return 0;
}
```

O dado **pertence a classe**, e não aos objetos. Sendo assim, **podemos** acessar os dados estáticos mesmo sem nenhum objeto instanciado da classe existir na memória.

Utilizando o membro estático

- Como utilizar o dado estático criado para dar um identificador único para cada conteúdo ministrado?

ConteudoMinistrado.hpp

```
#ifndef CONTEUDO_MINISTRADO_HPP
#define CONTEUDO_MINISTRADO_HPP

#include<string>

class ConteudoMinistrado{
        public:
                ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo);

                std::string& getDescricao();
                unsigned short getCargaHorariaConteudo();
                unsigned int getId();
        private:
                static unsigned int proxId;

                std::string descricao;
                unsigned short cargaHorariaConteudo;
                unsigned int id;
};
#endif
```

Deixamos o dado estático privado

Vai representar o id do objeto

UDESC

ConteudoMinistrado.cpp

```
#include "ConteudoMinistrado.hpp"

unsigned int ConteudoMinistrado::proxId{0};

ConteudoMinistrado::ConteudoMinistrado(std::string descricao, unsigned short cargaHorariaConteudo):
        descricao(descricao), cargaHorariaConteudo(cargaHorariaConteudo){
                this->id = ConteudoMinistrado::proxId;
                ConteudoMinistrado::proxId++;
}

std::string& ConteudoMinistrado::getDescricao(){
        return descricao;
}

unsigned int ConteudoMinistrado::getId(){
        return id;
}

//...
```

Damos um id para o objeto atual no construtor, e atualizamos o dado estático para cada objeto construído.

Teste você mesmo

```cpp
#include <iostream>
#include <string>
#include <list>


#include "Pessoa.hpp"
#include "Disciplina.hpp"
#include "SalaAula.hpp"
#include "ConteudoMinistrado.hpp"

int main(){
	ConteudoMinistrado c1{"Ponteiros", 4};
	ConteudoMinistrado c2{"Referencias", 2};


	std::cout << c1.getId() << std::endl;
	std::cout << c2.getId() << std::endl;



	return 0;
}
```

UDESC

Atenção

- A atualização do dado estático no construtor funciona assumindo que temos **apenas um processo ou thread** sendo executado em nosso programa
  - Programas com múltiplos *processos* ou *threads*, que são comuns em arquiteturas web (mesmo que você não esteja “consciente” disso) vão causar problemas de concorrência
    - Como podemos resolver?

Atenção

- A atualização do dado estático no construtor funciona assumindo que temos **apenas um processo ou thread** sendo executado em nosso programa
  - Programas com múltiplos *processos* ou *threads*, que são comuns em arquiteturas web (mesmo que você não esteja “consciente” disso) vão causar problemas de concorrência
    - Problemas podem ser resolvidos com *mutexes* ou *semáforos*, por exemplo
    - Veja detalhes na disciplina de Sistemas Operacionais

Adicionando em Disciplina

- Vamos adicionar uma lista de conteúdos ministrados na disciplina
  - A classe disciplina terá uma função que criará os conteúdos automaticamente

Disciplina.hpp

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

		void adicionarConteudoMinistrado(std::string conteudo, unsigned short cargaHorariaConteudo);
		void imprimirConteudosMinistrados();
	private:
		std::string nome;
		unsigned short int cargaHoraria;
		SalaAula* salaAula;

		Pessoa* professor;
		std::list<Pessoa*> alunos;
		std::list<ConteudoMinistrado*> conteudos;
};
#endif
```

UDESC

Disciplina.cpp

```
//...

void Disciplina::adicionarConteudoMinistrado(std::string conteudo, unsigned short cargaHorariaConteudo){
	this->conteudos.push_back(new ConteudoMinistrado{conteudo, cargaHorariaConteudo});
}

void Disciplina::imprimirConteudosMinistrados(){
	std::list<ConteudoMinistrado*>::iterator it;
	for(it = conteudos.begin(); it!=conteudos.end(); it++){
		std::cout << "Id: " << (*it)->getId() << std::endl
			<< "Conteudo: " << (*it)->getDescricao() << std::endl
			<< "Carga: " << (*it)->getCargaHorariaConteudo() << std::endl << std::endl;
	}
}
```

main.cpp

```cpp
#include <iostream>

#include "Pessoa.hpp"
#include "Disciplina.hpp"
#include "ConteudoMinistrado.hpp"

int main(){
	Pessoa prof1{"João", 40};
	Disciplina dis1{"C++"};
	dis1.setProfessor(&prof1);

	dis1.adicionarConteudoMinistrado("Ponteiros", 4);
	dis1.adicionarConteudoMinistrado("Referencias", 2);

	dis1.imprimirConteudosMinistrados();

	return 0;
}
```

Mais utilizações

- Dados estáticos são comumente utilizados para representar constantes em nossos programas
  - Algo que fazíamos via #define em C por exemplo
  - Mas temos maior flexibilidade e a compilação é mais simples
  - Comumente fazemos isso em conjunto com o modificador *const* (veremos em detalhes no futuro)

Exercícios

1. Complete a classe *Disciplina*
   - Adicione uma função *removerConteudoMinistrado(unsigned long id)*
   - Adicione uma função que retorna a lista de conteúdos ministrados
   - Apague da memória os conteúdos ministrados (que foram alocados dinamicamente) antes de terminar o programa
     - Você pode, por exemplo, criar uma função em Disciplina chamada “limparConteudos” que libera a memória, e chamar essa função antes de terminar o programa

2. Atualize o diagrama de classes para conter as novas classes e relações criadas
   Um dado estático é representado com um sublinhado no diagrama de classes
   Selecione o dado no StarUML, e marque a opção “*isStatic*”

3. Na classe retângulo (solicitada em aulas passadas) crie uma função membro que retorna quantos retângulos já foram criados no programa.

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.