Funções membro e modificadores de Acesso

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Classe Pessoa

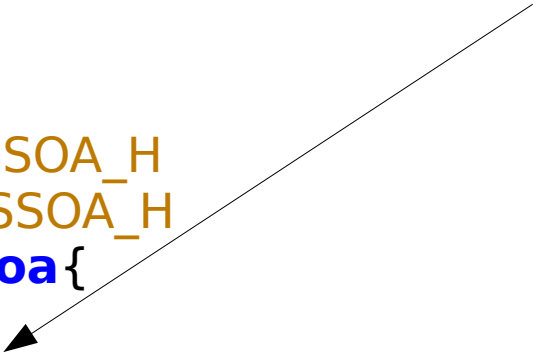
- Continuando com a classe pessoa
- Quais as funções fazem sentido ser executadas apenas para objetos do tipo Pessoa?
  - Em outras palavras, quais as funções membro?

Classe Pessoa

- Se utilizarmos o exemplo feito em C, a única função membro óbvia que temos é a validarCPF
- Vamos adicionar a função membro no header
  - Somente seu protótipo (assinatura)
- Como é o protótipo da função validarCPF?

Pessoa.hpp

C++ aceita o tipo primitivo bool (Booleano)



```
#ifndef PESSOA_H
#define PESSOA_H
class Pessoa{
    public:
        bool validarCPF(unsigned long cpfTeste);

        char nome[50];
        unsigned long cpf;
        unsigned char idade;
};

#endif
```

Pessoa.cpp

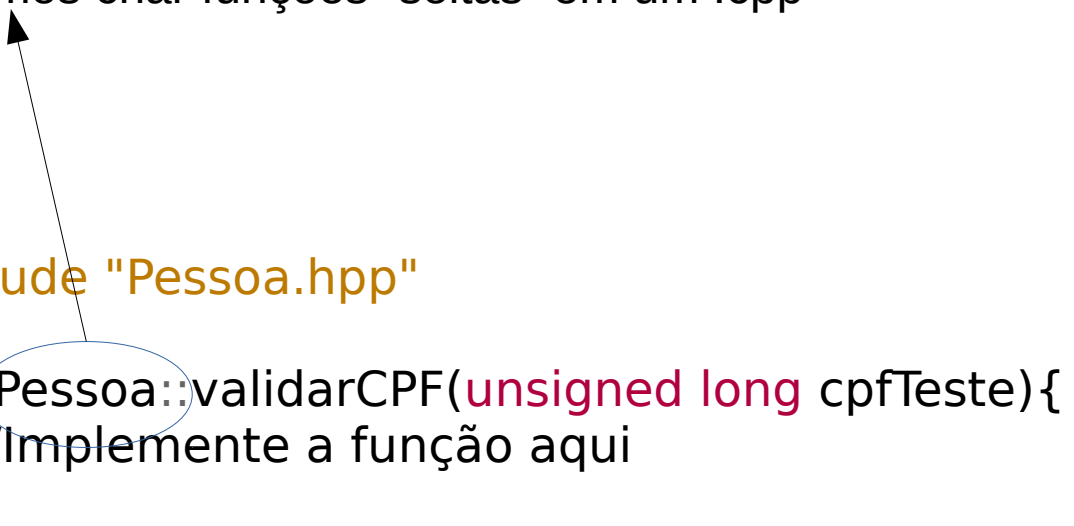
- A implementação completa da função membro se dá no .cpp
- Edite o arquivo Pessoa.cpp e adicione

```
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
      //Implemente a função aqui
}
```

Pessoa.cpp

Esse trecho indica que a função implementada é da classe Pessoa. É necessário já que um .hpp pode ter mais de uma classe (ex.: via classes aninhadas), e também podemos criar funções “soltas” em um .cpp



```
#include "Pessoa.hpp"

bool Pessoa::validarCPF(unsigned long cpfTeste){
    //Implemente a função aqui
}
```

ValidarCPF

- Agora podemos utilizar a função membro para validar CPFs.
- Para acessar uma função membro, **precisamos de um objeto instanciado**.

No main

```
#include<iostream>
#include"Pessoa.hpp"


int main(){
	Pessoa p1;
	std::cin >> p1.cpf;
	if(p1.validarCPF(p1.cpf) == true){
		std::cout << "CPF Valido: " << p1.cpf << std::endl;
	}else{
		std::cout << "CPF Invalido: " << p1.cpf << std::endl;
	}



	return 0;
}
```

UDESC

Discussão

- Assuma que todo cpf precisa ser validado
- Disponibilizamos a classe Pessoa pronta, para que outras pessoas possam utilizar
- Quais erros um programador (que não foi o que desenvolveu a classe Pessoa) pode cometer?

Discussão

- Assuma que todo cpf precisa ser validado
- Disponibilizamos a classe Pessoa pronta, para que outras pessoas possam utilizar
- Quais erros um programador (que não foi o que desenvolveu a classe Pessoa) pode cometer?
  - O principal deles no momento, é esquecer de validar o CPF antes de carregar o valor para um objeto do tipo pessoa

Discussão

- Até o momento, a orientação a objetos nos deu uma forma de organizar melhor nosso programa
  - Mas grande parte do que fizemos (senão tudo) poderia ser feito em C, se criássemos as structs corretamente, e separássemos tudo em arquivos .h
- Vamos melhorar nossa orientação a objetos em C++

Public

- O modificador de acesso **public** (veja no header de Pessoa) diz que podemos acessar as funções e dados membro a partir de qualquer local

```
#ifndef PESSOA_H
#define PESSOA_H
class Pessoa{
    public:
        bool validarCPF(unsigned long cpfTeste);

        char nome[50];
        unsigned long cpf;
        unsigned char idade;
};

#endif
```

Public

- Geralmente os dados membro não devem ser públicos
  - Os dados membro representam o estado interno de um objeto, e não desejamos que qualquer um realize alterações
  - Um dado membro público cria problemas similares a variáveis globais
    - Similares, mas não tão graves
    - Uma variável global é uma única variável compartilhada entre todo o programa
    - Um dado membro público pode ser acessado em qualquer parte do programa, mas cada objeto possui sua própria cópia do dado membro

Private

- Dados e funções membro marcadas como **private** são acessíveis apenas internamente na classe
  - Ou seja, somente as funções membro tem acesso a dados e outras funções membro marcadas como **private**
- Modifique o modificador de acesso de public para private na classe Pessoa
  - Dê um **make clean** para limpar os arquivos .o já compilados
  - Compile o código usando **make**
  - O que aconteceu?

Private

- O que aconteceu?
  - Temos um erro de compilação
  - Não podemos acessar os membros privados no main
- Mas desejamos acessar a idade do usuário, por exemplo
  - Ideias de como fazer isso mantendo os dados membro privados?

Gets e Sets

- Podemos criar funções membro públicas para dar acesso aos dados privados
- Funções membro que dão acesso possuem nomes especiais: gets e sets
  - get → Serve para retornar o valor de um dado privado
  - set →Serve para alterar o valor de um dado privado

Gets e Sets - Nomenclatura

- Os gets e sets devem possuir os seguintes nomes
  - tipoVariavel getNomeVariavel();
  - void setNomeVariavel(tipoVariavel novoValor);
- Por exemplo, para a variável *unsigned char idade* teremos
  - unsigned char getIdade();
  - void setIdade(unsigned char novaIdade);

Pessoa.hpp

```
#ifndef PESSOA_H
#define PESSOA_H
class Pessoa{
      public:
            unsigned char getIdade();
            void setIdade(unsigned char novaIdade);

      private:
            bool validarCPF(unsigned long cpfTeste);

            char nome[50];
            unsigned long cpf;
            unsigned char idade;
};

#endif
```

Pessoa.cpp

```cpp
#include "Pessoa.hpp"

unsigned char Pessoa::getIdade(){
    return idade;
}

void Pessoa::setIdade(unsigned char novaIdade){
    idade =  novaIdade;
}

bool Pessoa::validarCPF(unsigned long cpfTeste){
    //Implementação aqui ...
}
```

Usando no Main

```
#include<iostream>
#include"Pessoa.hpp"


int main(){
	Pessoa p1;
	int idade;
	std::cin >> idade;
	p1.setIdade(idade);

	std::cout << (unsigned short int)p1.getIdade() << std::endl;

	return 0;
}
```

UDESC

Exercício

- Faça os gets e sets para nome e cpf
  - 5 minutos!

Gets e Sets

- Quais vantagens de usarmos gets e sets?
- Quais desvantagens de usarmos gets e sets?

Gets e Sets

- Quais vantagens de usarmos gets e sets?
  - Podemos ter maior controle sobre os dados
    - Exemplo: se um dado não pode ser alterado, ele vai conter apenas get
  - Podemos realizar validações nos dados
  - Podemos mudar o comportamento da classe, sem modificar as assinaturas das funções membro (protótipos)
    - Isso vai ficar mais claro quando estudarmos sobrecarga de funções membro e polimorfismo
  - Podemos esconder detalhes da estrutura interna da classe
- Quais desvantagens de usarmos gets e sets?
  - Overhead
    - Chamar uma função é mais caro do que simplesmente alterar a variável

Gets e Sets

- Quais desvantagens de usarmos gets e sets?
  - Overhead
    - Chamar uma função é mais caro do que simplesmente alterar a variável
    - Podemos reduzir (ou até eliminar) isso através de funções inline
      - Veremos na próxima aula
    - Mesmo funções inline podem criar problemas, por conta do aumento na pressão na memória cache de dados
    - Por conta disso, algumas APIs de alto desempenho, como o OpenCV, preferem não utilizar gets e sets nas suas variáveis membro
      - As variáveis são públicas
      - Isso gera uma porção de problemas difíceis de resolver do ponto de vista da engenharia de software e O.O.
      - Mas elimina problemas de desempenho
    - Essa é a beleza do C++, você tem toda flexibilidade para fazer o que quiser, dependendo do problema que tem em mãos
      - Mas você precisa ter consciência das implicações geradas por suas decisões de design.

Gets e Sets

- Para tornar a classe Pessoa “autossuficiente”, como corrigir o problema do programador esquecer de validar o CPF?

Gets e Sets

- Para tornar a classe Pessoa “autossuficiente”, como corrigir o problema do programador esquecer de validar o CPF?

```
bool Pessoa::setCpf(unsigned long novoCpf){
    if(validarCPF(novoCpf)){
        cpf = novoCpf;
        return true;
    }
    return false;
}
```

Gets e Sets

- Note que a função “validarCPF” continua privada
  - O programador que importar a classe Pessoa nem vai saber que essa função existe
  - Mas agora a classe Pessoa é capaz rejeitar cpfs inválidos

```
bool Pessoa::setCpf(unsigned long novoCpf){
    if(validarCPF(novoCpf)){
        cpf = novoCpf;
        return true;
    }
    return false;
}
```

Gets e Sets

- Ainda temos um problema
  - O set agora tem um retorno (bool)
    - Isso é no mínimo estranho
  - Força o programador a colocar um if para verificar se o set deu certo
  - sets **não devem** possuir retorno
- Podemos corrigir isso via exceções
  - Veremos futuramente na disciplina

```
bool Pessoa::setCpf(unsigned long novoCpf){
    if(validarCPF(novoCpf)){
        cpf = novoCpf;
        return true;
    }
    return false;
}
```

Escondendo a estrutura interna da classe

- Internamente representamos a idade como um unsigned char
  - Economiza memória
  - Uma pessoa não vai ter mais que 255 anos
- Mas isso pode soar estranho para alguém que vai usar nossa classe
  - Podemos esconder esse detalhe dos programadores
  - A idade ainda é armazenada como *unsigned char*, mas os get e sets podem receber e enviar os dados como *unsigned short int*

Escondendo a estrutura interna da classe

Pessoa.hpp

```
#ifndef PESSOA_H
#define PESSOA_H

#include<string>

class Pessoa{
	public:
		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
```

Pessoa.cpp

```
//...

unsigned short int Pessoa::getIdade(){
	return idade;
}


void Pessoa::setIdade(unsigned short int novaIdade){
	//você pode ainda incluir validações aqui
	idade = (unsigned char)novaIdade;
}

//...
```

UDESC

Para a próxima aula

1)Modifique a classe retângulo solicitada na aula passada
- Adicione os conceitos que você aprendeu nessa aula
- Adicione um método membro para retornar o perímetro do retângulo
- Adicione um método membro para retornar a área do retângulo

2)Pesquise pela classe String em C++ (biblioteca string). Substitua o tipo da variável membro nome de vetor de caracteres para String.

Para ler uma string com espaços, utilize a função global getline

Está na biblioteca string, e no espaço de nomes std

Exemplo de uso

```
std::getline(std::cin, nome);
```

3)Leia sobre modelos anêmicos