"Bad programmers worry about the code. Good programmers worry about data structures and their relationships" (Linus Torvalds).

Modificadores de Acesso

DSBD

UFPR

Relembrando da aula passada

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

int main(){
   Pessoa p1;
   int idade;
   std::cout << "Nome: ";
   std::cin >> p1.nome;
   std::cout << "Idade: ";
   std::cin >> idade;
   p1.idade = idade;
   bool valido = false;
   while(!valido){
       std::cout << "CPF: ";
       std::cin >> p1.cpf;
       valido = p1.validarCPF(p1.cpf);
   }
   std::cout << "Dados da pessoa: " << p1.nome << "\t"
       << (unsigned short int)p1.idade << "\t" << p1.cpf <<
std::endl;

   return 0;
}
```

String

A **classe** `string` da biblioteca padrão é comumente usada para quando precisamos de vetores de caracteres.

Prático, já que os objetos dessa classe lidam com diversas complexidades automaticamente.

Ex.: lidam automaticamente com cópias.

Internamente, a classe string mantém um vetor de caracteres.

Para usar, basta incluir string via `include`, e lembrar que a classe está no espaço de nomes `std`.

Pessoa com String

```
#ifndef PESSOA_H
#define PESSOA_H

#include <string>

class Pessoa{
   public:
       bool validarCPF(unsigned long cpfTeste);

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

Discussão

Todo cpf precisa ser validado.

Disponibilizamos a classe Pessoa pronta, que pode ser usada em qualquer programa.

Quais erros um programador pode cometer?

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

int main(){
   Pessoa p1;
   int idade;
   std::cout << "Nome: ";
   std::cin >> p1.nome;
   std::cout << "Idade: ";
   std::cin >> idade;
   p1.idade = idade;
   bool valido = false;
   while(!valido){
       std::cout << "CPF: ";
       std::cin >> p1.cpf;
       valido = p1.validarCPF(p1.cpf);
   }
   std::cout << "Dados da pessoa: " << p1.nome << "\t"
       << (unsigned short int)p1.idade << "\t" << p1.cpf <<
std::endl;

   return 0;
}
```

Discussão

O erro mais óbvio no momento, é esquecer de validar o CPF antes de carregar o valor para um objeto do tipo pessoa.

Discussão

O erro mais óbvio no momento, é esquecer de validar o CPF antes de carregar o valor para um objeto do tipo pessoa.

Até o momento, a orientação a objetos nos deu uma forma de organizar melhor o programa.

Mas grande parte do que fizemos (senão tudo) poderia ser feito em C, criando-se structs corretamente, e separando em arquivos .h.

Vamos melhorar isso!

Public

O modificador de acesso `public` permite o acesso a funções e dados membro a partir de qualquer local.

```
#ifndef PESSOA_H
#define PESSOA_H

#include <string>

class Pessoa{
   public:
       bool validarCPF(unsigned long cpfTeste);

       std::string nome;
       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.

Não desejamos que qualquer um realize alterações.

Public

Geralmente os dados membro **não** devem ser públicos.

Os dados membro representam o estado interno de um objeto.

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 no objeto**.

Somente as funções membro da classe têm acesso a membros marcados como private.

Modifique o modificador de acesso de `public` para `private` na classe `Pessoa`.

Faça um `make clean` para limpar os arquivos .o já compilados.

Compile novamente o código usando `make`.

O que aconteceu?

Private

O que aconteceu?

- Diversos erros 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?

Funções de acesso

Podemos criar **funções membro públicas para dar acesso aos dados privados.**

Funções membro que dão acesso possuem nomes especiais: *getters* e *setters*.

get ⟶ Serve para retornar o valor de um dado privado.
Geralmente uma cópia do valor original.

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);
```

Convenção de Deitel, P. J., Deitel, H. M.. C++ how to Program: Pearson. 2017.

Por exemplo, para a variável unsigned long cpf teremos:

```
unsigned char getCpf();

void setCpf(unsigned long novoCpf);
```

Gets e Sets - Nomenclatura

Importante: esse será o padrão seguido na disciplina, mas não é o único existente.

Esse padrão é popular também em linguagens como Java e C#.

Alguns frameworks de grande porte, como o OpenCV, tentam evitar ao máximo o uso de getters e setters, e muitas vezes expõem diretamente os dados membro.

Getters e Setters nem sempre são populares...

Segundo Bjarne Stroustrup e Herb Sutter.

“Getters e setters triviais não adicionam valor semântico. O item poderia simplesmente ser público. Você poderia considerar sua classe como uma struct - ou seja, um grupo de variáveis públicas sem comportamento e sem funções membro ... Considere se faria alguma diferença além da sintaxe se o getter e setter fosse substituído pelo acesso público ao membro”.

Veja detalhes em isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rh-get

Exemplo

Pessoa.hpp

```
#ifndef PESSOA_H
#define PESSOA_H

#include<string>

class Pessoa{
   public:
       unsigned long getCpf();
       void setCpf(unsigned long novoCpf);

   private:
       bool validarCPF(unsigned long cpfTeste);

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

Pessoa.cpp

```
#include "Pessoa.hpp"

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

void Pessoa::setCpf(unsigned long novoCpf){
   cpf = novoCpf;
}

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

Exemplo

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

int main(){
   Pessoa p1;
   unsigned long cpfLido;
   std::cout << "CPF: ";
   std::cin >> cpfLido;
   p1.setCpf(cpfLido);

   std::cout << p1.getCpf() << std::endl;
   return 0;
}
```

Faça você mesmo

Faça os gets e sets para nome e idade.

5 minutos!

Resultado

Pessoa.hpp

```
#ifndef PESSOA_H
#define PESSOA_H

#include<string>

class Pessoa{
   public:
       unsigned long getCpf();
       void setCpf(unsigned long novoCpf);

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

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

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

Pessoa.cpp

```
#include "Pessoa.hpp"

#include <iostream>

unsigned long Pessoa::getCpf(){
   //retorna uma cópia do cpf
   return cpf;
}
void Pessoa::setCpf(unsigned long novoCpf){
   cpf = novoCpf;
}
std::string Pessoa::getNome(){
   return nome;
}
void Pessoa::setNome(std::string novoNome){
   nome = novoNome;
}
unsigned char Pessoa::getIdade(){
   return idade;
}
void Pessoa::setIdade(unsigned char novaIdade){
   idade = novaIdade;
}
bool Pessoa::validarCPF(unsigned long cpfTeste){
      //…
}
```

Resultado

main.cpp

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

#include"Pessoa.hpp"

int main(){
   Pessoa p1;
   unsigned long cpfLido;
   std::string nomeLido;
   unsigned short int idade;

   std::cout << "Nome: ";
   std::cin >> nomeLido;
   p1.setNome(nomeLido);

   std::cout << "Idade: ";
   std::cin >> idade;
   p1.setIdade((unsigned char)idade);

   std::cout << "CPF: ";
   std::cin >> cpfLido;
   p1.setCpf(cpfLido);

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

Vantagens e Desvantagens

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

Vantagens

- Maior controle sobre os dados.
  - Exemplo: se um dado não pode ser alterado, ele vai conter apenas get.
- Validação dos dados.
- Podemos modificar algum comportamento sem modificar as assinaturas das funções membro.
  - Isso vai ficar mais claro quando estudarmos sobrecarga de funções membro e polimorfismo.
- Podemos esconder detalhes da estrutura interna da classe.

Desvantagens

- Overhead.
  - Chamar uma função é mais caro do que simplesmente alterar a variável.
- Podemos mitigar o overhead através de funções inline.
  - Estude sobre funções inline (e note que é difícil ganhar do compilador).
  - Mesmo funções inline podem criar problemas, por conta do aumento na pressão na memória cache.
  - Por isso algumas APIs de alto desempenho, como o OpenCV, preferem não utilizar gets e sets nas suas variáveis membro.
    - As variáveis são públicas.
    - Gera uma porção de problemas difíceis de resolver do ponto de vista da engenharia de software e O.O.
    - Mas elimina problemas de desempenho.

Desvantagens

- Overhead.
  - Chamar uma função é mais caro do que simplesmente alterar a variável.
- Podemos mitigar o overhead através de funções inline.
  - Estude sobre funções inline (e note que é difícil ganhar do compilador).
  - Mesmo funções inline podem criar problemas, por conta do aumento na pressão na memória cache.
  - Por isso algumas APIs de alto desempenho, como o OpenCV, preferem não utilizar gets e sets nas suas variáveis membro.
    - As variáveis são públicas.
    - Gera uma porção de problemas difíceis de resolver do ponto de vista da engenharia de software e O.O.
    - Mas elimina problemas de desempenho.
- Em C++ você flexibilidade para fazer o que quiser, dependendo do problema que tem em mãos.
- Você precisa ter consciência das implicações geradas por suas decisões de design.

Validando o CPF

Como corrigir o problema do programador esquecer de validar o CPF?

Validando o CPF

Note que a função `validarCPF` continua privada.

O programador que importar a classe `Pessoa` não vai usá-la diretamente.

Mas agora a classe Pessoa é capaz de rejeitar cpfs inválidos.

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

Validando o CPF

Ainda temos um problema **grave**.

O set coloca zero no cpf, indicando que a pessoa não possui um cpf válido no momento.

Isso é no mínimo estranho.

Força o programador a colocar um if para verificar se o cpf do usuário é válido no momento, e o programador precisa saber que o "número mágico" 0 indica um cpf inválido.

Podemos corrigir isso via **exceções**.

Veremos futuramente.

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

Escondendo detalhes internos

Internamente representamos a idade como um unsigned char.

- Economizar memória.
- Uma pessoa não vai ter mais que 255 anos.

Mas isso pode soar estranho para alguém que vai usar a classe.

Podemos esconder esse detalhe.

- A idade ainda é armazenada como unsigned char.
- O get e o set podem receber e enviar os dados como unsigned short int.

Resultado

Pessoa.hpp

```
#ifndef PESSOA_H
#define PESSOA_H

#include<string>

class Pessoa{
   public:
       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
```

Pessoa.cpp

```
//...

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

void Pessoa::setIdade(unsigned short int novaIdade){
   if(novaIdade < 120)
       idade = (unsigned char)novaIdade;
   else
       idade = 0; //indicar erro
}

//...
```

Exercícios

1. Modifique a classe retângulo solicitada na aula passada.
    a. Adicione os conceitos que você aprendeu na aula de hoje.
    b. Adicione uma função membro para:
        i. Retornar o perímetro do retângulo.
        ii. Retornar a área do retângulo.
2. Pesquise sobre a classe string em C++ (biblioteca string).
    a. Dica: Para ler uma string com espaços, utilize a função global getline que está na biblioteca string, e no espaço de nomes std.
        Obs.: para proteger de buffer overflows, talvez seja melhor usar std::istream::getline e ler para um vetor de chars, e depois criar uma string a partir do vetor.

    Exemplo de uso std::getline(std::cin, nome)

3. Leia sobre modelos anêmicos.

Licença