


ORIENTAÇÃO A OBJETOS

Definição de estrutura

- Estruturas
 - Agregam outros tipos de dados em uma única estrutura
- ```
struct horario {
 int hora;
 int minuto;
 int segundo;
};
```
- 

## Definição de estrutura

- Estrutura auto-referenciável
  - Membros não podem ser instâncias de estruturas internas
  - Podem existir ponteiros para estruturas internas
    - Usados em listas encadeadas, filas, pilhas e árvores
- Definição
  - Cria novos tipos de dados usados p/ declarar variáveis
  - Exemplos:
    - `horario tObject;`
    - `horario tArray[ 10 ];`
    - `horario *tPtr;`
    - `horario &tRef = tObject;`

## Acessando Membros

- Operadores
  - Ponto (.) p/ estruturas e membros de classes
  - Seta (->) p/ estruturas e membros como ponteiros para objetos
  - Exemplo:

```
cout << tObject.hora;
OU
tPtr = &tObject;
cout << tPtr->hora;
```
  - `tPtr->hora` é o mesmo que `(*tPtr).hora`
    - Precisa de parênteses
      - \* tem menor precedência que .

## Tipo horario com struct

- Default: estruturas passadas por valor
  - Passe estruturas por referência
    - Evite trabalhos extras do sistema com cópias de estruturas
- C-style
  - Sem "interface"
    - Se a implementação é alterada, todos os programas que utilizam aquela `struct` devem se adaptar às mudanças
  - Não se pode imprimir como um todo
    - Deve-se imprimir membro por membro
  - Não se pode comparar estruturas
    - Deve-se comparar membro a membro

## Tipo horario com struct

- Exercício
  - Criar uma estrutura HORARIO
  - Criar funções para:
    - Inicializar um horário
    - Configurar (SET) um horário
    - Retornar (GET) um horário
    - Imprimir um horário na tela

## Tipo horario com struct

- Exercício
  - Criar uma estrutura HORARIO
  - Criar funções para:
    - Inicializar um horário
    - Configurar (SET) um horário
    - Retornar (GET) um horário
    - Imprimir um horário na tela

PROBLEMAS?

## Programação Orientada a Objetos

- Definições
  - Um **objeto** é uma entidade (concreta ou abstrata) do mundo real que tem uma identidade.
  - Em programação, cria-se uma **classe** (um molde, um carimbo) que possibilitará a criação de objetos.
    - Objetos serão instâncias das classes.
    - Objetos diferentes podem representar a mesma coisa, mas cada um será único.

## Programação Orientada a Objetos

- Um *objeto tem características e pode realizar ações.*
  - Exemplo: carrinho de controle remoto
    - Características (atributos)
      - marca, cor; velocidade máxima; peso; está ligado ou não; pilha carregada ou não; ...
    - Ações (métodos)
      - ir para frente; ir para trás; ligar ou desligar...

| Carrinho   |                                            |
|------------|--------------------------------------------|
| Atributos: | -marca<br>-cor<br>-ligado/desligado<br>... |
| Métodos:   | -ir pra frente<br>-ligar<br>...            |

## Programação Orientada a Objetos

- Algumas características de POO
  - Proteção;
  - Integridade dos dados;
  - Herança;
  - Generalização de operações;
  - Polimorfismo.

## Proteção e Integridade dos Dados

- Em um comprimido (cápsula), o conteúdo (remédio) está protegido do contato com o meio ambiente.
  - Em algumas definições de dicionário, encapsular significa proteger em uma cápsula.
- Encapsulamento
  - os dados de um objeto devem ser protegidos.



## Proteção e Integridade dos Dados

- Um relógio, por exemplo, deve marcar horas entre 0 e 23. Permitir que um relógio marque horas além disso é errado
- Um Ar condicionado não deve aceitar temperaturas muito altas ou muito baixas (além do limite de seu "hardware")

## Introdução

- Object-oriented programming (OOP)
  - Encapsula dados (atributos) e funções (comportamentos) em pacotes => classes
- Informações escondidas
  - Objetos se comunicam através de interfaces bem definidas
  - Detalhes de implementação ficam escondidos dentro das classes
    - Exemplo:televisor
      - Mudar valores em capacitores?
      - Volume acima do suportado?
      - Escolher frequências manualmente pelo "hardware"?

## Proteção e Integridade dos Dados

Classe CarroCR

- PROTEGIDO
  - COR
  - VELOCIDADE MÁXIMA
  - PESO
  - LIGADO/DESLIGADO
- LIVRE ACESSO
  - PRA FRENTE
  - PRA TRAS
  - LIGAR
  - DESLIGAR
- As informações protegidas não podem ser acessadas de qualquer lugar. Somente os métodos que estiverem na área de livre acesso poderão modificar as informações protegidas.

## Introdução

- Tipos definidos pelo usuário : classes
  - Dados (membros)
  - Funções (métodos)
  - Similares a carimbos – reutilizáveis
  - Instância de uma classe: objeto

## Implementando horario com uma class

- Classes
  - Modelam objetos
    - Atributos, características (dados)
    - Comportamentos (funções ou métodos)
  - Palavra-chave **class**
  - Funções Membro
    - Métodos
- Tipos de acesso
  - **public:**
    - Acessíveis sempre que o objeto estiver no escopo
  - **private:**
    - Acesso permitido somente para membros da classe
  - **protected:**

## Classes

- Construtor
  - Função especial
    - Inicializa dados
    - Mesmo nome da classe
  - Chamada quando o objeto inicializa
  - Vários construtores
    - Sobrecarga
  - Sem tipo de retorno

## A classe horario

```
1 class horario {
2
3 public:
4 horario();
5 void sethora(int, int, int); // set hora, minuto, segundo
6 void printUniversal();
7 void printStandard();
8
9 private:
10 int hora; // 0 - 23 (formato de 0 a 24 horas)
11 int minuto; // 0 - 59
12 int segundo; // 0 - 59
13
14 }; // fim da classe horario * note o ponto e vírgula
```

Protótipos de funções

Definição em public (acesso aberto).

Palavra-chave **class**

Constructor tem o mesmo nome que a classe, **horario**, e não tem tipo de retorno.

Definição termina com ponto e vírgula

## Classes

- Objetos da classe
  - Depois da definição
    - Nome (da classe) é um novo tipo
  - Exemplo:

Nome da classe se torna um novo tipo.

```
horario por_do_sol; // objeto do tipo horario
horario vet_horario[5]; // array de objetos horario
vet_horario[3]. printUniversal;
horario *ptrTohorario; // ponteiro p/ obj horario
horario &jantar = por_do_sol; // referencia
```

## Classes

- Funções definidas fora da classe
  - Operador de resolução de escopo (::)
    - Associa membro à classe
    - Identifica funções de uma classe particular
    - Classes diferentes podem ter funções c/ mesmo nome
  - Formato para definir funções-membro

```
ReturnType ClassName::MemberFunctionName(
){
 ...
}
```

    - Não muda quer a função seja **public** ou **private**
- Funções dentro da própria classe
  - Não precisam de operador de resolução de escopo

## Classes

- Destrutores
  - Também têm o mesmo nome da classe
    - ~ precede o nome
  - Sem argumentos
  - Não pode ser sobrecarregado
  - Faz a “limpeza da casa”

## Classes

- Operadores de acesso
  - Idênticos àqueles de estruturas
  - Ponto (.)
    - Objeto
    - Referência a objeto
  - Seta (->)
    - Ponteiros

## Funções de acesso e funções de utilidade

- Acesso
  - **public**
  - Leitura/escrita de dados
  - Funções para verificação de condições
- Utilidade (ajuda)
  - **private**
  - Fornecem suporte às funções **public**
  - Não são feitas para uso direto

## Inicialização de Objetos: Constructors

- Construtores
  - Inicializam membros
  - Mesmo nome das classes
  - Sem tipo de retorno
  - Existem os padrões (fornecidos pelo compilador – sem parâmetros) e os declarados pelo compilador.
  - Se declarar um, o compilador não lhe fornecerá o padrão.
- Inicializadores
  - Arguments passados para os construtores

```
Class-type ObjectName(value1,value2,...);
```

## Usando argumentos default com construtores

- Construtores
  - Podem especificar argumentos default
  - Construtores default
    - Todos os argumentos são defaults
  - OU
  - Explicita que não precisa de argumentos
  - Pode ser invocado sem argumentos

## Exemplo

```
1
2 // programa exemplo
3 #include <iostream>
4
5 using std::cout;
6
7 #include <iomanip>
8
9 using std::setfill;
10 using std::setw;
11
12 // inclui definição da classe horario no arquivo horario.h
13 #include "horario.h"
14
15 // Construtor inicializa cada membro com zero
16 // assegura que todos os objetos iniciam em 0
17 horario::horario(int hr, int min, int sec)
18 {
19 sethorario(hr, min, sec); // valida e ajusta horário
20 } // fim do construtor
21
22
```

Construtor chama  
sethorario p<sup>a</sup> validar  
valores.

## Continuação

```
23 // função sethorario
24 // valida valores e transforma os inválidos em zero
25 void horario::sethorario(int h, int m, int s)
26 {
27 hora = (h >= 0 && h < 24) ? h : 0;
28 minuto = (m >= 0 && m < 60) ? m : 0;
29 segundo = (s >= 0 && s < 60) ? s : 0;
30 }
31 // fim da sethorario
32
33 // imprime horário no formato universal
34 void horario::printUniversal()
35 {
36 cout << setfill('0') << setw(2) << hora << ":";
37 << setw(2) << minuto << ":";
38 << setw(2) << segundo;
39 }
40 // fim da função
41
```

## Continuação

```
42 // imprime horário no formato padrão
43 void horario::printStandard()
44 {
45 cout << ((hora == 0 || hora == 12) ? 12 : hora % 12)
46 << ":" << setfill('0') << setw(2) << minuto
47 << ":" << setw(2) << segundo
48 << (hora < 12 ? " AM" : " PM");
49 }
50 // fim da função
```

## Mais exemplo

```
...
10
11 int main()
12 {
13 horario t1; // todos os argumentos são default
14 horario t2(2); // default somente minutos e segundos
15 horario t3(21, 34); // segundo é default
16 horario t4(12, 25, 42); // todos os valores especificados
17 horario t5(27, 74, 99); // todos os valores inválidos
18
19 cout << "Construído com:\n\n";
20 << "todos os argumentos default:\n ";
21 t1.printUniversal(); // 00:00:00
22 cout << "\n ";
23 t1.printStandard(); // 12:00:00 AM
24
```

Inicializa t1  
usando argumentos  
default.

Inicializa t5 c/ valores  
inválidos; Todos os inválidos  
são transformados para 0.

## Destrutores

- Destrutores
  - Funções especiais
  - Mesmo nome das classes
    - Precedidos com (~)
  - Sem argumentos
  - Sem valores de retorno
  - Não podem ser sobrecarregados
  - Responsáveis pela "limpeza da casa"
  - Se não houver destrutor explícito
    - Compilador cria destrutores vazios.

## Quando construtores e destrutores são chamados

- Construtores e destrutores
  - Chamados “implicitamente” pelo compilador
- Ordem das chamadas
  - Depende da ordem de execução
  - Geralmente, destrutores são acionados na ordem reversa dos construtores

## Usando *Set* e *Get*

- Funções *Set*
  - Validam antes de modificar dados **private**
  - Podem notificar em caso de dados inválidos
- Funções *Get*
  - Retornam os valores
  - Controlam o formato dos dados de retorno

## Mais sobre classes

- Atribuição de objetos
  - Operador de atribuição (=)
    - Pode atribuir um objeto a outro do mesmo tipo
    - Default: memberwise assignment
      - Cada elemento à direita é atribuído individualmente ao elemento à esquerda
- Passando e retornando objetos
  - Objetos passados como argumentos de funções
  - Objetos retornados de funções
  - Default: passagem-por-valor
    - Cópia do objeto
      - Construtor de cópia
        - » Copia valores originais em novos objetos

## Exemplos de Classes

- Lâmpada
  - Tensão e potência
- Ar condicionado
  - Velocidade (1, 2, 3)
  - Temperatura (15 - 30)
  - Ligado/desligado