ORIENTAÇÃO A OBJETOS

SISTEMAS DE INFORMAÇÃO

DR. EDNALDO B. PIZZOLATO

Tópicos

- Definição de estrutura
- Acessando membros de estruturas
- O tipo "horario" com "struct"
- Implementando "horario" com "class"
- Escopo de classe
- Acesso a membros da classe
- Separando interface de implementação

Tópicos

- Controlando acesso aos membros
- Funções de acesso e funções de utilidade
- Inicialização de objetos: construtores
- Usando argumentos "default" c/ construtores
- Destrutores
- Chamada de construtores e destrutores
- Usando funções *Set* e *Get*

RELEMBRANDO...

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

Tipos definidos pelo usuário : classes

- Dados (membros)
- Funções (métodos)
- Similares a carimbos – reutilizáveis
- Instância de uma classe: objeto

RELEMBRANDO...

Estruturas

- Agregam outros tipos de dados em uma única estrutura

```
struct horario {
        int hora;
        int minuto;
        int segundo;
};
```

Nomes em estruturas

- Na mesma **`struct`**: nomes devem ser únicos
- Em diferentes **`structs`**: podem ser iguais

RELEMBRANDO...

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 x;`
  - `horario Array_horario[ 10 ];`
  - `horario *Ptr_horario;`

RELEMBRANDO...

Acesso aos membros (operadores)

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

```
cout << x.hora; // objeto simples
```

ou

```
cout << horarioPtr->hora; //ponteiro
```

- **`horarioPtr->hora`** é o mesmo que

`(*horarioPtr).hora`

  - Precisa de parênteses
    - `*` tem menor precedência que `.`

IMPLEMENTANDO HORARIO COM CLASSE

Classes

- Modelam objetos
  - Atributos, características (dados)
  - Comportamentos (funções ou métodos)
- Palavra-chave `class`
- Funções Membro
  - Métodos

IMPLEMENTANDO HORARIO COM CLASSE

Tipos de acesso

- **`public:`**
  - Acessíveis sempre que o objeto estiver no escopo
- **`private:`**
  - Acesso permitido somente para membros da classe
- **`protected:`**
  - Utilizado em herança (visto mais adiante)

IMPLEMENTANDO HORARIO COM CLASSE

O CONSTRUTOR

- Função especial
  - Inicializa dados
  - Mesmo nome da classe
- Chamado quando o objeto inicializa
- Vários construtores → sobrecarga
- Sem tipo de retorno

Programação Orientada a Objetos

```
1     class horario
2     {
3     public:
4        horario();                                  // constructor
5        void sethorario( int, int, int );           // set hora, minuto, segundo
6        void imprimir_24();        // imprime horário no formato 0 - 23h
7        void imprimir_12();         //imprime horário no formato 0 – 12h
8
9     private:
10       int hora;              // 0 - 23 (24-horas)
11       int minuto;            // 0 - 59
12       int segundo;           // 0 - 59
13
14    }; // fim da classe – não esquecer ;
```

Objetos da classe

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

```
horario acordar;   // objeto da classe
horario x[20]; // array de objetos

x[10].imprime_24(); // imprimindo...
```

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 (métodos) c/ mesmo nome

Funções definidas fora da classe

- Formato para definir funções-membro

```
tipo NomeClasse::NomeMétodo( ){
  …
}
```

- Não muda quer a função seja **`public`** ou **`private`**

Funções da própria classe

- Não precisam de operador de resolução de escopo

Destrutores

- Também têm o mesmo nome da classe
  - ~ precede o nome
- Sem argumentos
- Não podem ser sobrecarregados
- Fazem a "limpeza da casa"

Vantagens de se usar classes

- Simplificam a programação
- Interfaces
  - Escondem implementações
- Reuso de Software
  - Composição (agregação)
    - Classes incluídas como membros de outras classes
  - Herança
    - Novas classes derivadas de outras
    - Especificações de casos mais gerais

Operadores de acesso

- Idênticos àqueles de e**`struturas`**
- Ponto (.)
  - Objeto
  - Referência a objeto
- Seta (->)
  - Ponteiros

Funções de Acesso

- **`public`**
- Leitura/escrita de dados

Funções de Utilidade (ajuda)

- **`private`**
- Fornecem suporte às funções **`public`**
- Não são feitas para uso direto

Construtores

- Inicializam membros
- Mesmo nome da classe
- 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

- Argumentos passados para os construtores

```
NOMECLASSE Objeto( valor1,valor2,…);
```

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:

```
// Construtor  inicializa  membros
// assegura que todos os atributos
// iniciam em um estado consistente
   horario::horario( int hr, int min, int seg )
   {
      acertahorario( hr, min, seg );
   }
 // fim do construtor
```

EXEMPLO:

```
//  Muita gente simplesmente faz:
//          hora = hr;          // Mas, e se hora for 200?
//          minuto = min;       // Mas, e se minuto for 5000?
//          segundo = seg;      // Mas, e se segundo for -4?
horario::horario( int hr, int min, int seg )
{
   acertahorario( hr, min, seg );
}
// fim do construtor
```

EXEMPLO:

Muita gente simplesmente faz:

```
hora = hr;          // Mas, e se hora for 200?
minuto = min;       // Mas, e se minuto for 5000?
segundo = seg;      // Mas, e se segundo for -4?
```

```
void horario::acertahorario( int h, int m, int s )
{
    hora = (h >= 0 && h <=23)?h:0;
    minuto = (m >= 0 && m <=59)?m:0;
    segundo = (s >=0 && s <=59)?s:0;
}
```

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 destrutor vazio.

Construtores e destrutores

- Chamados "implicitamente" pelo compilador

Ordem das chamadas

- Depende da ordem de execução
- Geralmente, destrutores são acionados na ordem reversa dos construtores

Funções Set

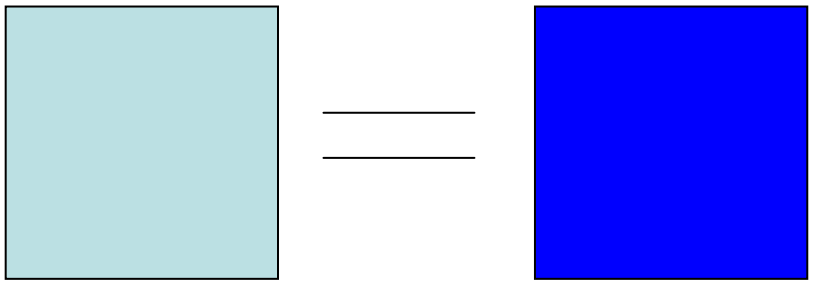
- Validam antes de modificar dados **`private`**
- Podem notificar em caso de dados inválidos

Funções Get

- "Query"
- Controlam o formato dos dados de retorno

Atribuição de objetos

- Operador de atribuição (=)
  - Pode atribuir um objeto a outro do mesmo tipo
  - Default: memberwise assignment
    - Cada elemento à direita é atribuido individuamente ao elemento à esquerda



Passando e retornando objetos

- Objetos são 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

ERROS COMUNS

O aluno quer passar os atributos para um método da classe.

Por que, se a classe já consegue ver o objeto?

```
horario x;
x.imprime_24h(hora,minuto,segundo);
```

ERROS COMUNS

O aluno quer atribuir um valor dentro do programa principal um atributo private.

Por que não usa uma interface?

```
horario x;
x.hora = 200;
```

ERROS COMUNS

O aluno esquece de indicar a qual classe pertence o método sendo implementado.

void acertahorario(int h, int m, int s)

Conclusões:

Structs não empacotam funções e variáveis em um único envelope.

Não há validação dos dados;

Não há garantia de integridade;

FIM