

Programação Orientada a Objetos

Giselle Lopes Ferrari Ronque

ferrari@eletrica.ufpr.br

Avaliações

- Provas
 - 27/03/2015
 - 15/05/2015
- Trabalhos
 - 12 e 19/06/2015
- Segunda chamada: 26/06/2015
- Exame: 10/07/2015

Regras

- Celular desligado.
- Respeito aos horários do início das aulas.
Faltas em excesso reprovam o aluno! (>25%)
Tolerância máxima de 15 minutos no início da primeira aula.
- Não comer/beber/fumar durante as aulas.
- Não navegar na internet/bate-papo.

Ementa

- Introdução à Orientação a Objetos
- Classes e Objetos
- Herança
- Polimorfismo
- Funções Virtuais
- Engenharia de Software
- Paradigma Cliente-Servidor;
- Paradigma Par-a-Par.

Bibliografia

- DEITEL, H. M.; DEITAL, P. J. C++ Como Programar. Porto Alegre: Bookman, 2001.
- AGUILAR, L. J. Programação em C++. São Paulo: McGraw-Hill, 2008.
- PRESSMAN, R. S. Engenharia de Software – Uma Abordagem Profissional. Porto Alegre: AMGH Editora Ltda.

Estruturas

REVISÃO

Estruturas

- Tipos complexos, modelados pelo programador;
- Declaração - Se **cadastro** é uma estrutura que vai conter um campo de inteiros para registrar idade, um campo de texto para nome, um campo real para salário e um campo caracter para sexo ('m'/'f'):

```
struct cadastro{  
    int idade;  
    char nome[30];  
    float salario;  
    char sexo;  
};
```

Estruturas

- Declaração - para criar um dado do tipo cadastro:
`cadastro funcionario, clientes[30];`

- Ou:

```
struct cadastro {  
    int idade;  
    char nome[30];  
    float salario;  
    char sexo;  
} funcionario, clientes[30];
```

Estruturas

- Se você precisa de apenas uma variável estrutura, o nome da estrutura não é necessário:

```
struct {  
    int idade;  
    char nome[31];  
    float salario;  
    char sexo;  
} funcionario;
```

Estruturas

- Para atribuir valores – nome da variável estrutura, seguida de ‘.’ e atribuição. Exs:

```
funcionario.sexo = 'm';
```

```
strcpy(clientes[11].nome, "José da Silva");
```

- Os campos das estruturas também são armazenados de forma contínua na memória, tal como os elementos das matrizes.

Estruturas

- Operações:

```
struct venda{  
    int pecas;  
    float preco;  
};
```

venda A = {20, 110.0}, B = {3, 16.5}, Total;

Total = A + B; // ERRADO

Total.pecas = A.pecas + B.pecas;

Total.preco = A.preco + B.preco;

Estruturas

- Estruturas podem ser aninhadas (campos são também estruturas).

```
struct data {  
    int dia, mes, ano;  
};
```

```
struct cadastro{  
    char Nome[30], Endereco[45];  
    int idade;  
    data nascimento;  
} funcionario;
```

```
funcionario.nascimento.ano = 1971;
```

Exercício 1

- Faça um programa que realize o cadastro de contas bancárias com as seguintes informações: número da conta, nome do cliente e saldo. O banco permitirá o cadastramento de apenas 15 contas e não pode haver mais de uma conta com o mesmo número. Crie o menu com as seguintes opções:
 - 1. Cadastrar contas**
 - 2. Visualizar todas as contas de um determinado cliente**
 - 3. Excluir a conta com menor saldo (supondo a não existência de saldo iguais.**
 - 4. Sair**

Passando Elementos de Estruturas para Funções

- Exemplo:

```
struct cadastro{  
    int idade;  
    float salario;  
    char sexo;  
} funcionario;
```

- Por valor

```
funcao1(funcionario.idade);  
funcao2(funcionario.salario);  
funcao3(funcionario.sexo);
```

- Por referência

```
funcao1(&funcionario.idade);  
funcao2(&funcionario.salario);  
funcao3(&funcionario.sexo);
```

Passando Estruturas Inteiras para Funções

- Por valor:

```
#include <stdio.h>
```

```
/*Variável Global*/
```

```
struct tipo{
```

```
    int a, b;
```

```
    char ch;
```

```
}
```

```
void f1(struct tipo t);
```

```
void main(){
```

```
    struct tipo X;
```

```
    X.a = 1000;
```

```
    f1(X);
```

```
}
```

```
void f1(struct tipo t){
```

```
    printf("%d", t.a);
```

```
}
```

Ponteiro para Estruturas

- Declaração
`struct cadastro*p;`
- Passagem por referência para funções:
 - É passado apenas o endereço da estrutura;
 - Referencia o argumento real em lugar de uma cópia.

Ponteiro para Estruturas

- Exemplo:

```
struct paciente{  
    float peso;  
    char nome[80];} A;
```

```
struct paciente*p;
```

```
p = &A;
```

```
p->peso = 100;
```

Exemplo

```
struct marca{
    int ano;
    char modelo[10];
};

void imprimir1(int a, char n[10]){
    printf("%d %s\n", a, n);}

void imprimir2(marca x){
    printf("%d %s\n", x.ano, x.modelo);}

void imprimir3(marca *x){
    printf("%d %s\n", x->ano, x->modelo);}
```

```
int main( ){
    struct marca renault[20], *p;
    renault[10].ano = 1950;
    strcpy(renault[10].modelo,
        "sander");
    p = &renault[10];
    imprimir1(renault[10].ano,
        renault[10].modelo);
    imprimir2(renault[10]);
    imprimir3(p);
    getchar();
    return 0;
}
```

Funções que retornam uma Estrutura

```
struct venda{  
    int pecas;  
    float preco;  
};
```

```
venda novavenda (void) {  
    venda X;  
    printf("Nova venda: \n");  
    printf("Insira o número de pecas: ");  
    scanf("%d", &X.pecas);  
    printf(" \nInsira o preco: ");  
    scanf("%f", &X.preco);  
    return X;  
};
```

```
void main (){  
    venda A, B;  
    A = novavenda();  
    B = novavenda();  
}
```

Exercício 2

Seja uma estrutura para descrever os carros de uma determinada revendedora, contendo os seguintes campos:

- marca: string de tamanho 15
- ano: inteiro
- cor: string de tamanho 10
- preço: real

a) Escrever a definição da estrutura carro.

b) Declarar o vetor vetcarros do tipo da estrutura definida acima, de tamanho 20 .

Crie um menu para:

c) Definir uma função para ler o vetor vetcarros.

d) Definir uma função que receba um preço e imprima os carros (marca, cor e ano) que tenham preço igual ou menor ao preço recebido.

e) Defina uma função que leia a marca de um carro e imprima as informações de todos os carros dessa marca (preço, ano e cor).

f) Defina uma função que leia uma marca, ano e cor e informe se existe ou não um carro com essas características. Se existir, informar o preço.