Universidade Federal de Ouro Preto - Campus Ouro Preto

Lívia Stéffanny de Sousa - 20.1.4029

BCC322 - Engenharia de Software I

LISTA III – REVISÃO DA LINGUAGEM C++

Ouro Preto
2022

Lívia Stéffanny de Sousa - 20.1.4029

LISTA III – REVISÃO DA LINGUAGEM C++

.

Terceira lista de exercícios da disciplina, Engenharia de software 1 do Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto.
Professor Orientador: Tiago G. S. Carneiro

Ouro Preto
2022

main1.cpp

1. Critique o código abaixo e aponte seus problemas com relação:

a. Flexibilidade da implementação: alterações em pequenos trechos do código não deveriam exigir recodificação de outras partes.

R: Em relação a flexibilidade da implementação o ideal seria separar as implementações do main.cpp, criaria um arquivo .cpp e .h para organizar o código.

b. Legibilidade do código: lendo o código é fácil entender os objetos e processos que ele representa. Qual é o seu objetivo?

R: O código não está legível, tanto nos nomes atribuídos às variáveis quanto na ideia principal de sua funcionalidade

2. Corrija os problemas identificados na questão 1.

R: Está respondida no arquivo main1.cpp.

3. Explique o código abaixo e cada uma das linhas de texto que formam sua saída.

R: A primeira linha do código é instanciado o construtor de Casa com a variável c1, e como parâmetro o valor de número 7. c2 é um ponteiro, do tipo Casa, que aponta para c1. c3, do tipo Casa, e recebe a referência de memória de c1. Em seguida, é exibido 'C1: ' com o valor de c1, é dado um 'getOrc' em c1. Em seguida, é dado um 'setOrc' em c1 passando como parâmetro o valor de número 3. Em seguida, é exibido 'C2: ' com o valor de c2, é dado um 'getOrc' em c2. Mas, como c2 é um ponteiro que aponta para c1, c2 exibe o mesmo valor que c1. Em seguida, é exibido 'C3: ' com o valor de c3, é dado um 'getOrc' em c3. Mas, como c3 tem a mesma referência de memória que c1, c3 exibe o mesmo valor que c1. Por fim, é exibido o endereçamento de memória de 'C2' e 'C3.

4. Quantas vezes o construtor da classe foi invocado? Por que?

R: Uma vez. Pois, somente o c1 foi instanciado. O c2 e c3 pega a referência de memória do c1.

main2.cpp

1. O código desse exercício é idêntico ao do exercício anterior.

a. Explique porque o método que sobrecarrega o operador "<<" precisou ser declarado como amigo ("friend") da classe Casa?

R: Quando declarado como friend é possível obter acesso aos membros 'private' e 'protected' de sua classe.

b. Explique o funcionamento do novo operador "<<".

R: Quando o operador '<<' sobrecarregado, do tipo ostrem, é possível exibir o objeto, de acordo com a implementação da função operator, sem precisar chamar um 'get' ou 'toString'.

2. Explique o código abaixo e cada uma das linhas de texto que formam sua saída.

R: A primeira linha do código é instanciado o construtor de Casa com a variável c1, e como parâmetro o valor de número 7. c2, do tipo Casa, e recebe a referência de memória de c1. Em seguida, é exibido 'C1: ' com o valor de c1, é dado um 'getOrc' em c1 e exibido 'C2: ' com o valor de c1, é dado um 'getOrc' em c2. Em seguida, é dado um 'setOrc' em c2 passando como

parâmetro o valor de número 3. Em seguida, é exibido ‘C1’ e ‘C2’, com sobrecarga do operator’<<’, com seus valores.

main3.cpp

1. A função abaixo cria nomes com base em um prefixo escolhido pelo programador. Por exemplo, para o parâmetro "Cliente" ela cria os nomes: Cliente0, Cliente1, Cliente2,...

R:

a) Para permanecer na memória durante a execução e não perder a contagem.

b) Pois, se ele for declarado como ‘char*’ vai ser necessário usar o ‘new’ para alocar ele na memória.

c) Pois, é necessário espaço para o ‘\0’.

d) Pois, assim ela só é alocada quando for utilizada. A variável global ficará alocada durante toda execução.

e) Pois, com isso ele vai alocar com o tamanho adequado.

2. Antes de executar o código abaixo, pressione "ctrl+alt+del" e ative o painel "desempenho" do gerenciador de tarefas do Windows. Execute o código e observe o consumo de memória registrado pelo sistema operacional. Explique porque a demanda por memória aumenta sempre que seu programa é executado. Altere a classe "Cliente" para corrigir essa falha na sua implementação. Obs: Não se esqueça de recompilar todo o código para que sua correção tenha efeito.

R: Faltou o destrutor.

3. Para resolver esta questão, antes, você deverá resolver a QUESTÃO 2. Antes de executar o código abaixo, pressione "ctrl+alt+del" e ative o painel "desempenho" do gerenciador de tarefas do Windows. Então, execute o código e observe que o erro de consumo indevido de memória retornou. Entretanto, sua causa agora é outra. Ou então aconteceu o pior, seu programa está abortando e não funciona mais devido ao erro gerado pela linha "cli1 = cli2". Comente esta linha e então observe o resultado do programa. Explique a saída do programa e a causa do erro? Sobrecarregue o operador "=" da classe "Cliente" e corrija o problema. Garanta que sua correção passe sem problemas pelo código: "cli1 = cli1", onde um objeto "Cliente" é atribuído a ele mesmo.

R: Faltou a sobrecarga do operador ‘=’.

4. Ao invés de declararmos uma variável global para mantermos a contagem do número de objetos da classe "Veiculo" carregados na memória, decidimos declarar a variável "cont" na própria classe "Veiculo". Qual é a vantagem dessa abordagem? Por que essa variável precisa ser declarada como "static" para que nosso código funcione? Experimente tirar o qualificador "static" e executar a nova versão do código para entender, por completo, seu funcionamento. Ao retirar o qualificador "static" da variável, vc perceberá que os métodos que fazem acesso a variável "cont" também precisarão deixar de ser "static".

R: É necessário ser declarada como static para não precisar acessar criando um objeto.

5. Por que o método "getCont()" pode ser invocado por meio do nome da classe, "Veiculo::getCont()", e não é preciso um objeto para invoca-lo?

R: Pois ele é um método público e estático.

6. Na classe "Funcionario" o método "reajustaSalario" foi declarado como um método "virtual". Observe o código abaixo e explique o por que. Para entender melhor o funcionamento desse código, retire o qualificador "virtual" do método "reajustaSalario", execute o programa novamente e observe sua saída.

R: Foi declarado como virtual para que as classes que herdam possam implementar de maneira que for necessária a elas. Classe Gerente e Mecânico, também possuem o método reajusteSalario, porém com outra implementação. É possível por causa do virtual.

7. O código a seguir faz a listagem dos carros da concessionária classificando-os como "biciclo" ou "quadriciclo". Repare que a constante "classe" que armazena a classificação do veículo, não foi declarada na classe "Veiculo", pois não se sabe a essa altura se o objeto a ser instanciado terá 2(duas) ou 4(quatro) rodas. Desta maneira, somente a classe "Moto" e "Automovel" possuem a constante "classe" para a classificação automática do veículo instanciado. Entretanto, é desejável que se possa invocar o método "getClasse()" a partir de objetos do tipo "Veiculo". Por esse motivo, a classe "Veiculo" também possui esse método na sua interface. Responda os seguintes itens:

R:

a) Não é possível criar um objeto de uma classe com método puramente virtual.

b) Pois é obrigatório para as classes que herdam da de Veículo.

c) Para o ponteiro e o objeto que eles apontam serem ambos constantes. Pois,

assim não podem ser modificados.

8. No código a seguir, a constante "MOTO" possui o tipo "Moto", assim como a variável "moto". Explique por que é possível invocar o método "getClasse()" por meio variáveis e não é possível invocá-lo através da constante. Corrija a definição do método "getClasse()" para que seja possível invocá-lo tanto por meio de uma variável quanto de uma constante.

R: É necessário adicionar ‘const’ para transformar o método em constante para poder ser invocado por uma instância de constante.

Exemplo:

virtual const char* const getClasse(void) const = 0;

const char* constgetClasse(void) const;