1)A programação modular é um exemplo de paradigma orientado a objetos pois lida com classes e objetos.Certo ou Errado.Justifique.

R:Correto.Embora a programação modular não utilize nomes como classe,objeto e métodos, podemos fazer uma analogia entre os elementos dela com os da programação orientada a objeto. Imagine a seguinte situação: Temos o módulo peça que possui uma estrutura com o mesmo nome, a qual contém cor e id. Este também possui diversas funções de acesso. Agora, imagine que declaramos 6 ponteiros para peça em um módulo distinto da mesma aplicação. Podemos considerar esses seis ponteiros como sendo instâncias da classe peça(objetos), os campos cor e id como sendo os atributos da classe e as funções de acesso como sendo os métodos da classe. Portanto, sim, de uma certa perspectiva a programação modular pode ser considerada uma programação orientada a objetos

2)Apresente o esquema de algoritmo do quickSort apontando ao menos um hotspot preenchido com parâmetros do tipo ponteiro para função.

R:

3)Faça a argumentação de corretude completa do algoritmo abaixo:

INICIO

Ind <— 1

ENQUANTO IND <= LL FAÇA

INICIO

ATUAL <— ELE[IND]

AUX <— IND - 1

ENQUANTO AUX >= 1 E ELE[AUX] > ATUAL

INICIO

ELE[AUX+1] <— ELE[AUX]

AUX <— AUX + 1

FIM

ELE[AUX + 1] <— ATUAL

IND <— IND + 1

FIM

FIM

R:

4)Transforme uma estrutura grafo em auto-verificável

R:

5)Gere quatro deturpações possíveis para a estrutura da questão 4.

R:Deturpações:

a)Torna ponteiro para lista de adjacências de algum vértice em nulo

b)Torna ponteiro para lista de adjacências de algum vértice em Lixo

c)Torna o ponteiro para o vetor de lista de adjacências em nulo

d)Faz com que o campo quantidade de arestas seja diferente do valor real de arestas

6)Apresente o código do verificador para a estrutura da questão 4.

R:

GRA\_Verificador(GRA\_tppGraph g) {

int i, qtdArestas = 0;

if(g->vetAdjs == NULL) {

CNT\_Contar(“erro-vetAdjsNulo”);

}

for(i=0;i<g->qtdVert;i++) {

if(g->vetAdjs[i] == NULL) {

CNT\_Contar(“erro-listaDeAdjsNula”);

}

if(g->vetAdjs[i] == LIXO) {

return GRA\_tpCondRetErroListaDeAdjsLixo;

}

qtdArestas += g->vetAdjs[i]->qtdNos;

}

if(qtdArestas != g->qtdArestas) {

return GRA\_tpCondRetErroQtdArestasErrada;

}

}

7)Distribua controladores de cobertura no verificador e informe o relatório final de contagem executando apenas as deturpações da questão 5.OBS:Deve ser inserido o menor número de controladores necessários para testar completamente o verificador pelo critério caixa aberta.

R:(Ainda incompleta)

GRA\_Verificador(GRA\_tppGraph g) {

int i,qtdArestas = 0;

if(g->vetAdjs == NULL) {

CNT\_Contar(“erro-vetAdjsNulo”);

return GRA\_tpCondRetErroVetAdjsNulo;

}

for(i=0;i<g->qtdVert;i++) {

if(g->vetAdjs[i] == NULL) {

return GRA\_tpCondRetErroListaDeAdjsNula;

CNT\_Contar(“erro-listaDeAdjsNula”);

}

if(g->vetAdjs[i] == LIXO) {

CNT\_Contar(“erro-listaDeAdjsApontaLixo”);

return GRA\_tpCondRetErroListaDeAdjsLixo;

}

qtdArestas += g->vetAdjs[i]->qtdNos;

}

if(qtdArestas != g->qtdArestas) {

CNT\_Contar(“erro-listaDeAdjsApontaLixo”);

return GRA\_tpCondRetErroQtdArestasErrada;

}

}

8)Apresente um exemplo de codigo em C utilizando trace de evolução.

R:

void loop(int n) {

int i;

for(i=0;i<n;i++) {

printf(“%d\n”,i); //trace de evolução

}

}

9)Explique a vantage da instrumentação em relação aos testes convencionais no que diz respeito ao esforço de diagnose.

R: Utilizando a instrumentação, vários erros que são extremamente difíceis de identificar e que estão fortemente relacionados com violações de assertivas estruturais, tornam-se bem mais fáceis de serem percebidos, pois com a instrumentação sempre estamos verificando se a estrutura instrumentada está de acordo com suas assertivas estruturais.

10)Qual é o objetivo de se definir a assertiva invariante de uma repetição?Apresente o código de uma assertiva invariante executável.

R: