1. Bibliotecas estáticas (.lib) são arquivos já previamente compilados (.obj) que estão ligadas a um programa (usuário/cliente) no momento de sua compilação, são necessárias no ato de compilar na medida que cada arquivo no programa necessita de uma “cópia” da biblioteca no tempo de compilação, dessa forma, quando o programa é compilado gera um executável independente de arquivos externos. Diferentemente, as bibliotecas dinâmicas (.dll), apesar de serem arquivos já compilados, existem independentemente do programa (usuário/cliente) criado, dessa forma, este programa passa a depender da biblioteca dinâmica como arquivo externo, ou seja, no tempo de execução do programa.

2.

3. O exemplo dado em sala era sobre campos de preenchimento obrigatório em formulários durante uma compra online, um exemplo diferente é a mensagem “ seu documento não foi salvo ainda, deseja sair mesmo assim ?” que aparece em alguns editores de texto online quanto tentamos sair e ele verifica que não salvamos o arquivo ainda.

4. A programação modular permite a criação de baselines de módulos fonte já aprovados; e com o desenvolvimento incremental é possível criar várias versões do programa com um todo, com uma versão sendo mais completa que a de antes. A cada versão, temos também uma atualização da baseline , com o aprimoramento individual, é possível substituir uma parte do programa por outra com as mesmas funcionalidades e com melhor desempenho, sem afetar o restante do programa. Então, podemos concluir que esses três conceitos juntos permitem o desenvolvimento de um módulo bem projetado , sendo os três conceitos um vantagem de programação modular .

5. Na coesão lógica, os elementos possuem funcionalidades relacionas mas não existe uma única definição bem delimitada para o conjunto de elementos. Na coesão funcional os elementos definem uma função bem delimitada. Um exemplo bem simples para os elementos seguindo uma coesão lógica é um módulo que apaga desenhos de uma tela independente dos formatos, apaga quadrados, apaga elipses,etc. Uma coesão funcional seria se os módulos estivessem separados por formato, um módulo apaga quadrados, outro módulo apaga elipses e assim vai. Nesse caso, a funcionalidade de cada módulo está bem definida

6. Certo

7. Um exemplo de acoplamento de baixa qualidade é uma função que recebe muitos parâmetros. Podemos solucionar isso passando uma struct.

8. A possibilidade de salvar o jogo

9. Existir a possibilidade de jogar duas partidas de gamão ao mesmo tempo, pois isso faz com que o sistema tenha que suportar dois tabuleiros.

10.

11. Verdadeiro, não é porque é genérico que é mal formulado. Um requisito precisa satisfazer ao que está especificando, e se um requisito genérico satisfizer o que o cliente procura e for bem formulado nesse sentido, sem ambiguidade, então está bem formulado.

12. A situação mais comum de definição sem declarar uma variável é a utilização, na linguagem C/C++, a utilização da função malloc presente na biblioteca stdlib. Esta função aloca um espaço de memória do tamanho em bytes retornando um ponteiro para esta região de memória, portanto, definindo mas não declarando.

13. Errado, pois um espaço de dados somente é definido quando existe um espaço de memória alocado(definir) e quando é declarado, ou possui um ou mais nomes de referência

14. É possível utilizar os comandos para pré-processador #ifdef, #ifndef, #else e #endif. Esses comandos permite que o pré-processador altere partes do código e/ou reutilize partes ignorando outras

Exemplo:

#ifdef \_\_EXEMPLE\_OWN\_\_

#define \_\_EXEMPLE\_EXT\_\_

#else

#define \_\_EXEMPLE\_EXT\_\_ extern

#endif

\_\_EXEMPLE\_EXT\_\_ int vector[3];

#ifdef \_\_EXEMPLE\_OWN\_\_

= {1,2,3};

#else

#endif

Neste caso, quando um módulo definir o \_\_EXEMPLE\_OWN\_\_ e incluir este .h ele estará criando uma variável para ser utilizada no módulo. Se não está definido \_\_EXEMPLE\_OWN\_\_ o módulo estará referenciando uma variável global vector definida em outro módulo.

15. Os ponteiros pra função no trabalho principalmente para passar o destrutor dos elementos de uma lista, pois esta ultima é generica, portanto não "sabe" destruir seus elementos, assim é nescessário passar o ponteiro pra função destrutor do elemtento para a lista, para essa poder destruir seus elementos corretamente.

16.

int search(estrutura,value, int (\*getmin)(estrutura), int(\*getmax)(esturutra), int (\*comparator)(value1, value2){

int min = getmin(estrutura);

int max = getmax(esturutra);

int meio;

while(min < max){

meio = (min+max)/2;

int comp = comparator(estrutura[meio],value);

if(comp == IGUAL) return TRUE;

else if(comp == MENOR){

max = meio -1;

}

else{

min = meio +1;

}

}

}

17. Errado. Uma função recursiva indireta pode não implementar uma chamada circular entre módulos, basta que todas as chamadas sejam de funções pertencentes de um mesmo módulo.

Exemplo: sejam as funções F1 e F2 do mesmo módulo, a chamda F1->F2->F1, a função F1 é uma chamada recusirva indireta, porém, como todas as funções são do mesmo módulo, ela não implementa uma chamada circular.

18. Essa chamada é uma chamada recursiva indireta, pois apesar de ser composta por uma chamda de função recursiva direta (F3->F3), a função F1 chama outras funções e termina em F1 novamente.

19. Errado. Uma função morta(não é chamada para nada) para uma arquitetura modular pode ser utilizada em alguma outra arquitetura, em algum outro acoplamento.