**Caderno Caio Graça Melo - 1621170**

**Aula 1 Modular (13/03/2019) -> Aula 4 (25/03/2019)**

**Introduçao**

**Vantagens da programação mudular:**

* Vencer barreiras de complexidade.
* Destribuir tarefas em um grupo. Saber particionar problemas.
* Reutilização de modulos/funções. A reutiização permite a redução do esforço ao longo do projeto, dependendo do quão necessária ela seja.
* Criação de um acervo de modulos reutilizáveis.
* Permite trabalhar com baselines de modulos já testados. A partir de criado o executável, a manutenção e aprimoramento é feita a partir dessa baseline.
* Desenvolvimento incremental. Adição de módulos a fim de aprimorar o projeto.
* Aprimoramento individual. Aprimoramento individual de cada modulo melhorando as partes do modulo que nao necessitem mudanças em outros modulos. Essencialmente são alterações de um modulo visando a melhora, sem mexer nos parametros de entrada e nem no retorno.
* Reduzir o tempo de compilação. Executando poucos módulos, o tempo de compilação é reduzido.

**Princípios de modularidade**

* Modulo:  
  - Física: unidade de compilação independente.  
  - Lógica: possui um unico conceito.
* Elementos de programa  
  - Hierarquia:  
   SISTEMA > PROGRAMA > MODULO > CLASSE > FUNÇÕES > BLOCOS DE CODIGO > LINHA  
  Podem ser: Bloco de codigo, fragmentos de texto de documentação, funções, figuras ou diagramas, sessões de documentação, tipos de dados, classes, componentes...  
  - Conceitos:  
   Artefato: algo elaborado durante um processo de desenvolvimento e que possua identidade propria. Artefato é o que consegue ser versionado no desenvolvimento de uma aplicação.  
   Construto: é uma versão de uma aplicação que pode ser executada mesmo que incompleta.
* Interface  
  - Formas de interface:  
   Mecanismo de trocas de dados, comandos e eventos entre elementos de programa. O modulo só faz sentindo se houver a conversa com outros modulos. A interface sempre ocorre entre eventos de mesmo nivel na hierarquia. Dois sistemas podem se relacionar através de arquivos, esses são um exemplo de interface. Outro modo de comunicação entre arquivos é uma função, essa é um metodo de comunicação entre modulos. O tipo de interface de blocos de codigo são as variaveis globais.  
  - Relacionamento cliente servidor:  
   Cliente chama função que acessa o servidor.  
   Caso especial: Callback. Cliente chama a função, o servidor responde solicitando mais dados e o cliente chama a função solicitando mais dados.   
  - Interface fornecida por terceiros:  
   Para a comunicação entre dois módulos, é necessário a criação de um .h para fazer a definição e "tradução" desses módulos. Um bom programador não faz código duplicado.  
   Definição : É quando um tipo utilizado em uma interface entre dois módulos não está definida em nenhum dos módulos de implementação, e sim em um módulo de definição comum aos dois.  
  -Interface em detalhe:  
   Sintaxe : Regras -> Float != Int  
   Semântica : Significado -> int idade != int numero da casa  
  -Exemplo de interface:  
   tpDadosAlunos\* obterAluno(int id)  
   - Interface esperada pelo cliente:  
   Ponteiro válido referenciando os dados de aluno ou null.  
   - Interface esperada pelo servidor:  
   Id válido, acesso aos dados de aluno (se o mesmo existir) (interface explícita).  
   - Interface esperada por ambos:  
   tpDadosAluno (interface fornecida por terceiros, interface explícita) -> assinatura ou protótipo.  
  OBS: Protocolo de uso:   
   Forma de se utilizar os itens que compoem uma interface para que esta possa ser operada corretamente. Deve estar na documentação de cada função.
* Processo de Desenvolvimento  
  
* Cliente -> M1.c  
  Servidor -> M2.c e M2.h  
  M1.h deve ser incluído na compilação do módulo M2.  
  .h -> Módulo de definição  
  .c -> Módulo de implementação.
* Módulo de Definição (.h)  
  - Especificação externa voltada para os programadores do módulo cliente.   
  - Protótipos ou assinaturas das funções de acesso.   
  - Declarações e códigos públicos ao módulo.
* Módulo de Implementação (.c)  
  - Especificação interna voltada para os programadores do módulo servidor.  
  - Protótipos ou assinaturas das funções internas.  
  - Declarações e códigos encapsulados no módulo.  
  - Códigos executáveis das funções.
* Tipo Abstrato de Dados (TAD)  
  É a estrutura encapsulada que somente é conhecida pelos clientes através das funções de acesso.  
  - Cabeça tem ponteiro para o primeiro elemento e um ponteiro pra corrente.  
  - Para criar uma lista, primeiramente é necessario criar uma caebeça. Após isso, criaremos os ponteiros das colunas 1 e 2, criando um nó entre elas e a cabeça.  
  **Dados os exercícios 1-4 da lista de exercícios.**
* Propriedades da Modularização:  
  - Encapsulamento  
  - Acoplamento  
  - Coesão
* Encapsulamento  
  Propriedade relacionada à proteção dos dados de um componente de forma que este possa ser utilizado sem perder suas caracteristicas basicas.  
  -Vantagens: Facilitar a manutenção pois tudo que está relacionado à estrutura encapsulada está dentro de um determinado módulo. Facilita a documentação pois tudo que se encontra documentado trata de um único conceito.  
  - Desvantagem: Exagero do encapsulamento de conceitos.  
  -Tipos de encapsulamento:  
   - de código: O código da função de acesso contido no módulode implementação e que não é vista pelo modo cliente. Outros exemplos de encapsulamento de código são: For, While, If, Código de uma Função.  
   -de variáveis: Variável static é encapsulada ou na classe (caso OO) ou no módulo (caso Modular). Local encapsulada em blocos de código. Private é encapsulada no objeto. Protected é global a todas as estruturas de herança  
   - de documentação:   
   - Interna: Documentação do (.c)  
   - Externa Documentação da interface (.h)  
   - de Uso: Documentação voltada para o usuário.
* Acoplamento  
  Propriedade relacionada à interface entre os módulos.  
  -Conector: É o item da interface.   
  Protótipos de função, arquivos, variáveis globais aos módulos são alguns exemplos de interface.  
  - Critérios de qualidade de acoplamento: O menor acoplamento possível é o melhor.  
   - Tamanho do conector. Ex: Função com 10 parâmetros x Função com 2 parâmetros.  
   - Quantidade de conectores. Colocar apenas o necessário à disponibilidade do modo cliente. Por exemplo, funções internas não tem necessidade de serem adicionadas ao .h .  
   - Complexidade do conector. Com a criação de um procolo de uso, a utilização de uma função "complexa" pode ser facilitada.
* Coesão  
  Propriedade relacionada ao grau de interdependência dos elementos que compõem o módulo (conceito).  
  - Níveis de coesão: A melhor coesão é aquela que o módulo trata de um único conceito.  
   - Incidental: Não há relação entre os vários conceitos inseridos no módulo. "É a pior coesão que existe".  
   - Lógica: Os elementos possuem uma relação lógica entre os conceitos de uma forma possivelmente genérica. Ex: Módulo que calcula.  
   - Temporal: Os elementos estão relacionados pela necessidade de serem utilizados dentro do mesmo período de tempo.  
   - Procedural: Os elementos estão relacionados pela necessidade de serem utilizados em um determinada ordem. Exemplo: .bat .  
   - Funcional: Os elementos estão relacionados por funcionalidades. Tudo relacionado a mesma funcionalidade está no mesmo módulo.  
   - Abstração de dados: Tipo mais forte de coesão. Módulo que só trata de um único conceito.  
  **Exercícios 5 a 7 da lista dados.**

**Lista de exercícios 1**1) Apresente as diferenças entre biblioteca estática e dinâmica.

R: A biblioteca dinâmica (dll) é junta ao programa quando o programa é executado enquanto a biblioteca estática (lib) é acoplada ao programa na hora da compilação. Quando um executável é gerado, se ele possuir uma dependência de alguma biblioteca dinâmica que não esteja no projeto, ele não irá funcionar. Já com a biblioteca estática isso não ocorrerá pois ela foi acoplada ao executável gerado na compilação. Algumas das outras diferenças entre as bibliotecas estática e dinâmica são o tamanho das aplicações. O tamanho da aplicação de uma biblioteca estática é geralmente maior que o executável da biblioteca dinâmica. Além disso, a biblioteca estática não permite algumas otimizações enquanto a biblioteca dinâmica permite otimizações globais.

2) Uma aplicação trabalha com um vetor de grafos criados com listas em que cada elemento do grafo armzaena uma matriz tridimensional de listas genéricas. Apresente os módulos e interfaces desta aplicação explicando os relacionamentos clientes-servidor, interface fornecida por terceiros e as principais funções de acesso.

3) Dê um exemplo de callback diferente do apresentado em sala.

4) Qual é a relação entre os conceitos de baseline, desenvolvimento incremental e apriomoramento individual?

R: Baseline é a base do projeto já executada e testada. A partir disso, são feitos o desenvolvimento incremental e aprimoramento individual. O primeiro é a adição de módulos com o intuito de aprimorar o projeto e o aprimoramento individual é a otimização de modulos sem alterar seus parâmetro e retornos, assim evitando alterações em outros módulos dependentes.

5) Explique a diferença entre coesão lógica e funcional. Apresente exemplos.

6) Um módulo com bom acoplamento pode ter uma coesão ruim. Certo/Errado/Depende/Tipo Assim. Justifique sua resposta.

7) Dê um exemplo de acoplamento de baixa qualidade.

**Aula 5 Modular (03/04/2019)**

**Especificação de Requisitos**

* Requisitos  
  - O que deve ser feito.  
  - Nunca como deve ser feito.
* Características dos Requisitos  
  -Curtos e diretos.  
  - Linguagem natural.
* Etapas da especificação  
  - Elicitação: Buscar informações para gerar uma documentação de requisitos.  
   -Técnicas:  
   - Entrevista.  
   - Brainstorm.  
   - Questionário.  
  - Documentação  
   - Requisitos genéricos.  
   - Requisitos específicos  
   - Contrato  
  - Verificação  
   - Análise se a documentação somente possui requisitos computáveis.  
   - Junto com a equipe técnica.  
  - Validação  
   - Cliente.
* Tipos de Requisito  
  - Funcional  
   - Regras que devem ser implementadas na aplicação relacionadas com o negócio.  
  - Não Funcionais  
   - Propriedades que a aplicação deve possuir e que não necessariamente estão relacionadas com o negócio.  
   a) Segurança  
   - Login e senha  
   b) Disponibilidade  
   - ex: 24/7  
   c) Backup  
   d) Velocidade  
   - Todas as consultas devem retornar o resultado no máximo em 3 segundos.  
  - Inverso  
   - O que a equipe se compromete a não fazer.
* Exemplos de Requisitos  
   a) Bem formulados:   
   -Para cada aluno deve ser cadastrado matricula e nome.  
   - O relatório de turmas deve ser disponibilizado no primeiro dia de matricula.  
   b) Mal formulados:  
   - A interface deve ser de fácil utilização  
   - O relatório apresenta seus dados mais necessários.

**Aula 6 (08/04/2019) de Modular**

**Modelagem de dados**

1 modelo gera n exemplos. O modelo bom é aquele que representa n exemplos.  
Notação: UML like.  
  


Exemplos:  
- Vetor  
- Lista simplesmente encadeada com cabeça  
- Lista duplamente encadeada genérica com cabeça  
- Árvore binária genérica com cabeça  
  
Assertivas Estruturais:  
- Lista:   
 Se pCorr->pAntes != null  
 pCorr->pAntes->pProx==pCorr  
 Se pCorr->pProx!=null  
 pCorr->pProx->pAntes==pCorr  
  
 Ponteiro de um nó de subarvore a esquerda nunca aponta para pai nem para nó de subarvore a direita.  
 OBS: T2 Modelo = modelo, assertivas e exemplo  
- Vetor de lista de árvore  
-Grafo: 4 tipos de lista diferentes:  
 - Vertice: aponta pra arestas  
 - Aresta: aponta pro vertice  
 -Vertices: aponta vertice  
 - Origens: aponta vertice

**Aula 7 Modular (15/04/2019)**

**Assertivas** Regras consideradas válidas ao executar um determinado ponto no programa. São utilizadas em:  
- Argumentação de corretude.  
- Instrumentação.

**Assertivas Estruturais** São regras que complementam um modelo de uma estrutura de dados.

**Assertivas de Entrada e Saída** Regra considarada válida antes da execução de comandos/ Regra considerada válida depois da execução de comandos.  
 Assertiva de Entrada válida --> Assertiva de Saída válida  
  
OBS: Função fake, retorna o valor esperado mesmo sem a execução apropriada da função.  
OBS2: As assertivas precisam estar corretas e completas.  
Exemplo:   
 --------> Assertiva de entrada  
 Excluir nó corrente intermediário de uma lista duplamente encadeada com cabeça.  
 --------> Assertiva de Saída  
  
 Assertiva de entrada: Lista existe e possui pelo menos 3 nós. Ponteiro corrente aponta para o nó intermediário que se quer excluir. Valem as assertivas estruturais da lista duplamente encadeada com cabeça.  
 Assertiva de saída: Nó foi excluido. Valem as assertivas estruturais da lista duplamente encadeada com cabeça. Ponteiro corrente aponta para o primeiro nó da lista.

**No trabalho 2, criar as assertivas de entrada e saída para todas as funções de acesso.**

**Mais exercícios da lista:**8) Dê um exemplo de requisito não funcional do seu trabalho do período.

R: Um requisito não funcional do trabalho do período é a condição de retorno de algumas funções, que servem pra checar se a função foi executada corretamente.  
  
9) Elabore um requisito não funcional que resulta em um requisito funcional.  
  
10) Apresente modelo, exemplo e assertivas de uma estrutura utilizada no armazenamento de um cubo mágico.  
  
11) É possível existir um requisito genérico bem formulado. Certo ou Errado? Justifique.

R: Errado pois o requisito genérico é muito abrangente, não focando muito nas especificidades do sistema.

**Aula 8 Modular (24/04/2019)**

**Implementação da Programação Modular**

**1) Espaço de Dados**- São áreas de armazenamento alocadas em um meio.  
- Possuem um tamanho.  
- Possuem um ou mais nomes de referência.  
Ex: A[j] -> j-ésimo elemento do vetor.  
 \*ptAux -> Espaço apontado por um ponteiro.  
 ptAux-> Espaço que contém um endereço.  
 ptroTabSimb\* ObterElemTabSimb (char\* ptSimbolo):  
 ObterElemTabSimb(char\*ptSimbolo)->Id. --> Subcampo da estrutura apontada pelo ponteiro retornado pela função ObterEleTabSimb.

**2) Tipos de Dados**- Determinam:  
 - Organização:  
 Como você lê o código binário.  
 - Codificação:  
 Regra semântica do código.  
 - Tamanho em bytes.  
 - Conjunto de valores permitidos.  
OBS: Um espaço de dados precisa estar associado a um tipo para que possa ser interpretado pelo programa desenvolvido em linguagem tipada.  
OBS2: Tipos de tipos:  
 - Tipos Computacionais:  
 O tipo de dados mais simples possível. Ex: int, char etc.  
 - Tipos Básicos:  
 Struct, Union, Enum, typedef.  
 - Tipos Abstratos de Dados  
 Função de acesso encapsulada num módulo

**3) Tipos Básicos**- Struct: Representa diversas informações.  
- Union: Representa vários tipos de dados simultaneamente em apenas um espaço.  
- Enum  
- Typedef.

**4) Declaração e Definição de Elementos**Definir:- Aloca espaço de dados  
 - Amarra espaço a um nome (binding)  
Declarar:  
 - Corresponde o espaço a valores de um determinado tipo.  
int x -> Declara e define.  
malloc -> so define.  
struct -> so declara.

**5) Implementação em C/C++** a) Declarações e definições de nomes globais exportados pelo módulo servidor  
 int a;  
 b) Declarações externas contidas no módulo cliente e que somente declaram o nome sem associa-lo.  
 extern int a;  
 c) Declarações definições de nomes globais encapsulados no módulo.  
 static int a;

**Aula 9 Modular (29/04/2019)**

**Pré-Processamento**

código fonte + intruções de pré processamento (#include) → pré processamento (executa os hashtags) → código fonte → compilador → código objeto

#define nome valor → substitui o nome pelo valor  
#include <nome arquivo> → copia o texto da biblioteca e coloca no seu código  
#if defined(nome) ou #ifdef nome  
#else ou #if !defined  
#end if  
#undef nome  
#if ! defined (exemp\_mod)  
#define exemp\_mod  
 corpo do .h  
#end if

exemp\_ext int vetor [7]  
#if defined (exemp\_own) = {1,2,3,4,5,6,7}  
#else  
#endif

**Exercícios Modular**

12) Apresente uma situacao de definicao sem declarar a variavel.

13) 'E possivel considerar que apenas declarar sem definir chega a definir um espaco de dados?. Certo/Errado/Talvez. Justifique

14)Como e possivel personalizar interfaces para modulos clientes sem duplicar codigos? Apresente exemplos.

**Aula 10 Modular (06/05/2019)**

**Estrutura de Funções**

**Paradigma**

**1) Forma de programar**  
 -procedural: programação estruturada  
 -orientada a objetos: programação orientada a objetos  
 -programação modular (mesmo utilizando umalinguagem não orientada a objetos)

**2) Estrutura de Funções**



**3) Estrutura de Chamadas**



- F4 → F4: chamada recursiva direta

- F9 → F8 → F3 → F5 → F9: chamada recursiva indireta

- F10: função morta (em outra aplicação ela pode ter utilidade)

- F8 → F3 → F5 → F6 → F7: dependência circular entre módulos

- Supondo: F6 → F7: arco de chamada

**4)Função**  
É uma porção auto-contida de código. Possui:  
 -um nome  
 -uma assinatura  
 -um ou mais (ponteiro de função) corpos de código

**5) Especificação da Função** -Objetivo (pode ser igual ao nome)  
 -Acoplamento (parâmetros e condições de retorno)  
 -Condições de acoplamento (assertivas de entrada e assertivas de saída)  
 -Interface com o usuário (mensagens, saídas em tela para o usuário)  
 -Requisitos (o que deve ser feito)  
 -Hipótese: são regras pré-definidas que assumem como válida uma determinada ação ocorrendo fora do escopo, evitando assim o desenvolvimento de códigos desnecessários.  
 -Restrições: são regras que limitam as escolhas das alternativas de desenvolvimento para uma determinada solução.

**6) Interfaces**  
 -conceitual → definição da interface da função sem preocupação com a implementação. Exemplo: inserirSimbolo( Tabela, Simbolo ) → Tabela, IdSimbolo, condRet  
 -física → implementação da conceitual. Exemplo:tpCondRet insSimb( tpSimb \* simbolo ) tabela → global static no módulo  
 -implícita (dados de interface diferentes de parâmetros e valores de retorno)

**7) Housekeeping**  
 Código responsável por liberar componentes e recursos alocados a programas ou funções ao terminar a execução.

**Aula 11 Modular (08/05/2018)**

**7) Repetição**



**8) Recursão**



**9) Estado  
-** Descritor de Estado: Variavel(eis) que definem o estado.  
 Ex: pesq seq -> ind  
 pesq binaria->inferior/superior  
- Estado: Valoração do descritor de estado.

**10) Esquema de Algoritmo**{  
 inf=obterLimInf(); // hotspot  
 sup=obterLimSup(); // hotspot  
 while(inf<=sup)  
 {  
 meio=(inf+sup)/2;  
 comp=comparar(valorProc, obterValor(meio));  
 if(comp==igual) {break;}  
 if(comp==menor) {}  
 else {inf=meio+1;}  
 }  
}

OBS: Esquemas de algortimo permitem encapsular a estrutura de dados. É correto, é incoompleto e precisa ser instanciado. Normalmente ocorrem em:  
- Poo  
- Frameworks  
Se esquema correto e hotspots com assertivas válidas então programa correto.

**11) Parâmetros do Tipo Ponteiro para Função**float areaQuad(float base, float altura)  
{ return base\*altura;}  
  
float areaTri(float base, float altura)  
{ return base\*altura/2;}  
  
int ProcessaArea(float valor1, float valor2, float (\*func)(float,float))  
{  
 printf("%f", func(valor2,valor2));  
}  
  
condRet=ProcessaArea(5,2,areaQuad);  
condRet=ProcessaArea(3,2,areaTri);

**Continuação Lista de Exercícios**15) Explique como funciona a utilização de ponteiro de função nos trabalhos do período.

16) Elabora um esquema de algortimo em que um dos hotspots é preenchido dinamicamente por um valor apontado por uma varivel ponteiro de função.

17) Toda chamada recursiva indireta é uma dependência circular entre modulos. Certo ou Errado? Justifique.

18) F1 -> F3 -> F3 -> F4 -> F1 ->é qual tipo de chamada?

19) Função morta é código inutil. Certo ou Errado? Justifique.