Programação Modular

Introdução:
Vantagens de programação modular

- Vencer barreiras de complexidade do trabalho
- Facilita o trabalho em grupo, após a divisão de tarefas no grupo.
- Re-uso
- Facilita a criação de um acervo (Diminui a quantidade de novos programas implementados)
- Desenvolvimento incremental
- Aprimoramento individual
- Facilita do administrador de baselines

Princípios de modularidade:

1) Módulo
    - Definição física: Unidade de compilação independente
    - Definição logica: Trata de um único conceito

2) Abstração de sistema
    - Abstrair e o processo de considerar apenas o que é necessário em uma situação e descartar com segurança o que não é necessário.
    - Níveis de abstração: Sistema > Programa > Módulos > funções > Bloco de código > linhas de código

      Obs: Conceito artefato: é um item com identidade própria criado dentro de um processo de desenvolvimento que pode ser versionado.

    - Construto (build): Algo para apresentar, artefato.

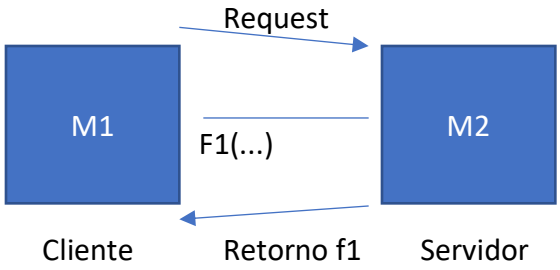
3) Interface

   Mecanismo de troca de {dados, estados, eventos} entre elementos de um mesmo nível de abstração.

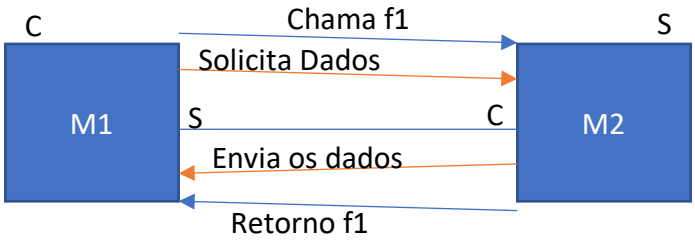
    a) Exemplo de Interface:
        - Arquivo (entre sistemas)

- Funções de acesso (entre módulos)
- Passagem de parâmetros
- Variáveis globais (entre blocos)

b) Relacionamento cliente-servidor:



Caso Especial: Callback

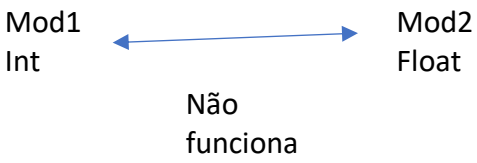


c) Interface fornecida por terceiros

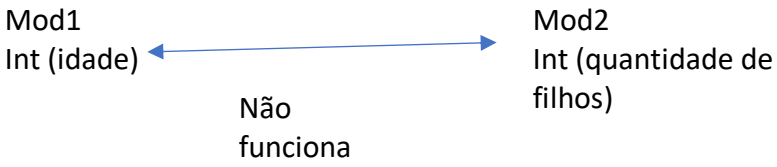
TpAluno.c←TpAluno.h→ TpAluno1.c

d) Interface em detalhe

- Sintaxe: Regra

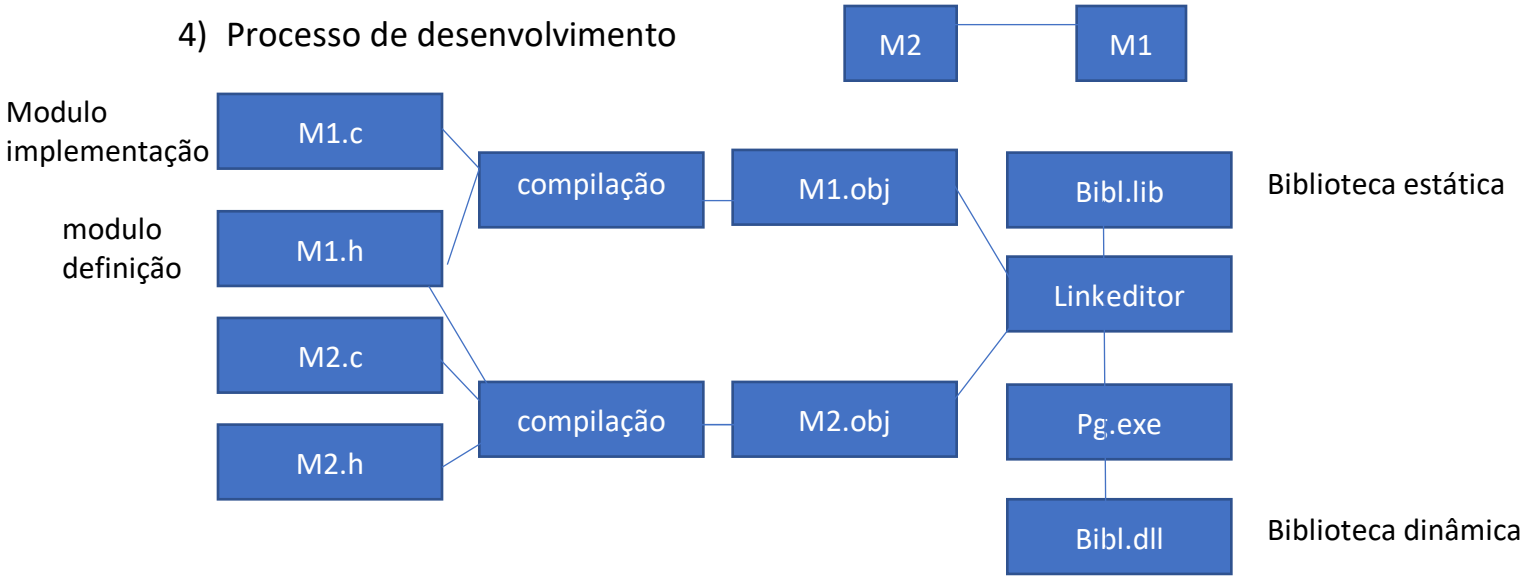


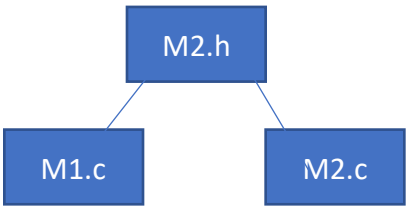
- Semântica: Significado



e) Análise de Interface
   - Tp Dados Aluno * ObterDadosAluno(int mat) → Protótipo ou assinatura de função de acesso
   - Interface esperada pelo cliente, ponteiro para dados válidos do aluno correto ou null.
   - Interface esperada pelo servidor, inteiro válido representando a matricula de um aluno.
   - Interface esperada por ambos
     TpDados Aluno

4) Processo de desenvolvimento





5) Bibliotecas estáticas e dinâmicas
   - Estática
     Vantagens:
     - Lib acoplada em tempo de linkediçao a aplicação executável

Desvantagens:

- Existe uma copia dessa biblioteca para cada executável na memoria que a utiliza

▪ Dinâmica

Desvantagens:

- A dll precisa estar na maquina para a aplicação funcionar

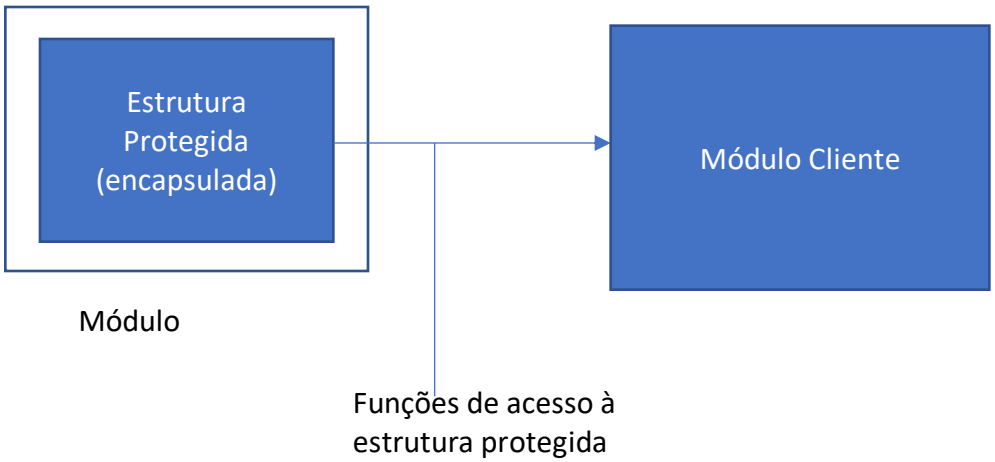
Vantagens:

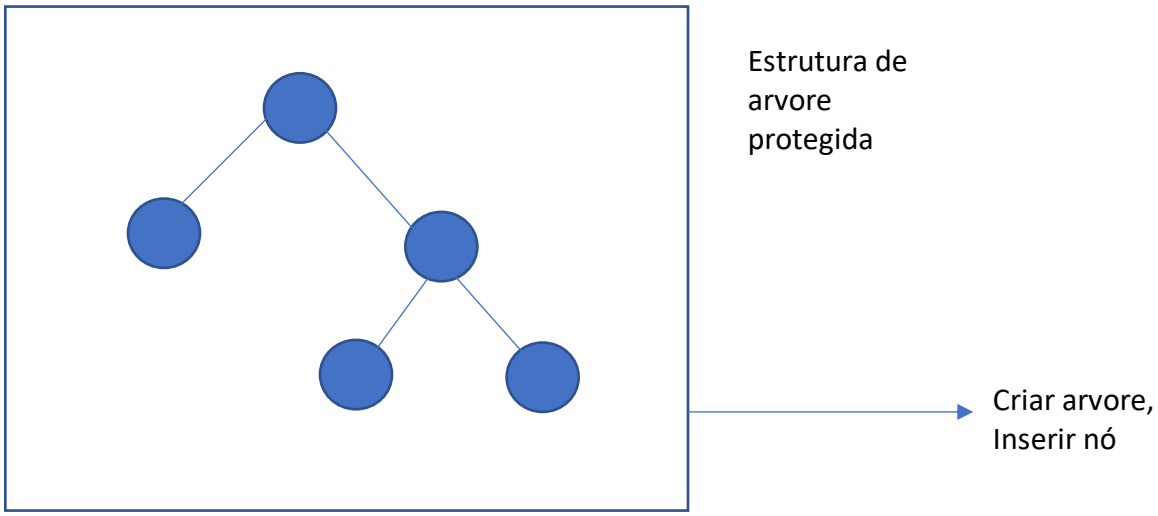
- Só tem uma instancia na memoria: só é carregada uma instancia na biblioteca dinâmica na memoria mesmo que varias aplicações à acessem.

6) Módulo de definição(.h)
   - Interface do modulo
   - Contem os protótipos das funções de acesso interfaces fornecidas por terceiros (ex: tpDadosAluno de item 3e)
   - Documentação voltada para o programador do modulo-cliente.

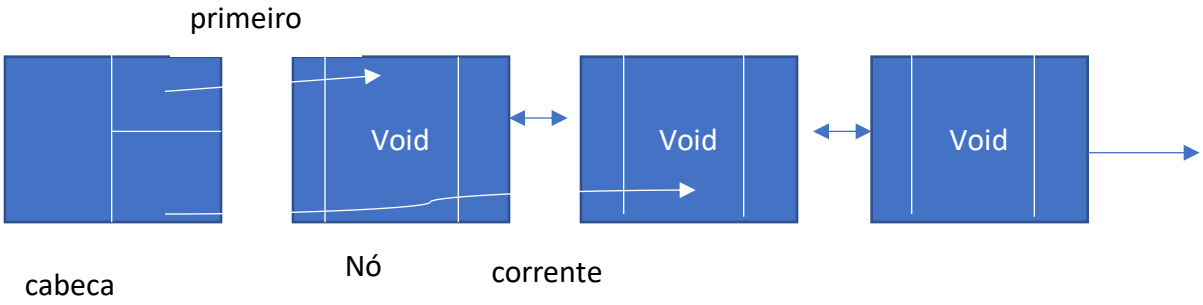
7) Modulo de implementação (.c)
   - Código das funções de acesso
   - Código e protótipos das funções internas
   - Variáveis internas ao modulo
   - Documentação voltada para o programador do modulo servidor

8) Tipo Abstrato de Dados





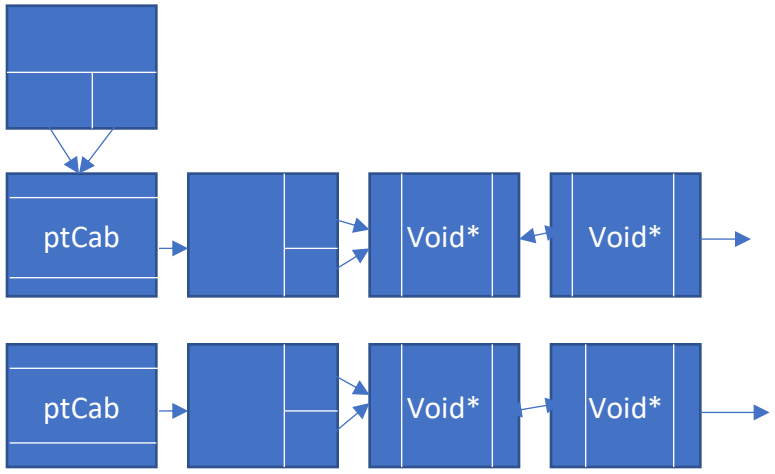
- TAD, é uma estrutura encapsulada em um modulo que somente é conhecido pelos módulos cliente através das funções de acesso disponibilizadas na interface.



No TAD, são usadas funções como:
CriarLista(ptCab *plista);
InserirNo(ptCab plista, void *item);

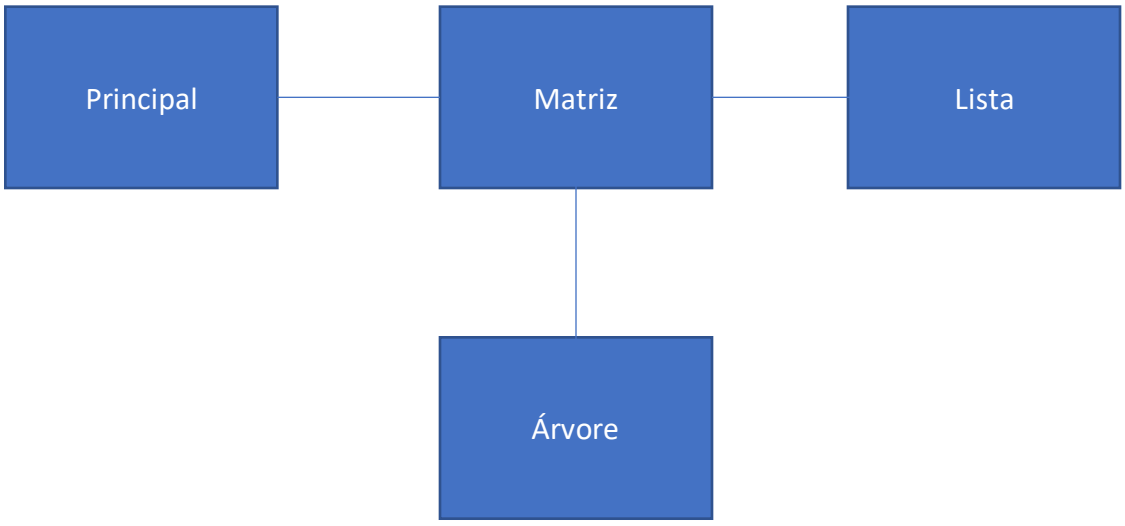
- Na primeira função e usado passagem de parâmetros por referencia, aonde o endereço de uma variável no modulo cliente terá seu valor alterado no modulo cliente.
- Na segunda função a passagem de parâmetros por valor, é acessado o valor da variável para uso interno, sem alterar seu valor.

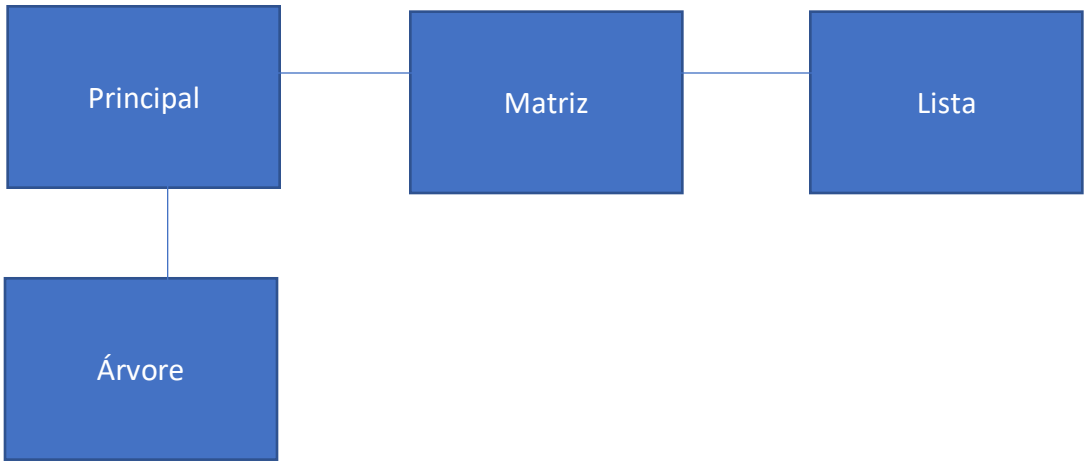
Para montar uma matriz 2x2 por meio de TAD lista:
Crialista(p1);
Crialista(p2);
inserirNo(p2,NULL);
inserirNo(p2,NULL);
inserirNo(p1,p2);
criaLista(p3);
inserirNo(p3,NULL);
inserirNo(p3,NULL);
inserirNo(p1,p3);



- Ao reciclar o código da lista na implementação do modulo matriz, o modulo principal não precisa fazer chamadas de funções de acesso do modulo lista, essas chamadas são feitas no modulo matriz.

  Na estrutura abaixo o modulo matriz necessita de lista em sua estrutura e também pode armazenar árvores em seus elementos. Porém o módulo principal não pode criar árvores fora da matriz.





Neste modelo é possível criar arvores e matrizes separadamente, e também matrizes de arvores. O modulo principal pode fazer isso.

9) Encapsulamento
   Propriedade relacionada com a a proteção dos elementos que compõem um modulo.
   Objetivo:

- Facilitar a manutenção
- Impedir a utilização indevida da estrutura de dados

Outros tipos de encapsulamento:

- Documentação interna – modulo impli.c
- Documentação externa – modulo def.c
- Documentação de uso – manual do usuário

De código:

- Blocos de código visíveis apenas
- Dentro do modulo
- Dentro de outro bloco de código (ex: conjunto de comandos dentro de um for)
- Código de uma função

De variáveis:

- Private(encapsulada no objeto), public, global, global static (modulo), protected (estrutura heranças), static (classe), local(bloco de código), etc

10) Acoplamento

- Propriedade relacionada com a interface entre os módulos
- Conector → item de interface

Ex: função de acesso

- Variável global

Critérios de qualidade:

- Quantidade de conectores

Necessidade X Suficiente
(tudo é útil?) (falta algo?)

- Tamanho do conector (ex: quantidade de parâmetros de uma função)

Complexidade do conector:

- Explicação em documentação
- Utiliza mnemônicos → nomes compatíveis

11) Coesão

Propriedade relacionada com o grau de interligação dos elementos que compõem um modulo:

Níveis de coesão:

- Incidental – pior coesão
- Logica – elementos logicamente relacionados
- Temporal – itens que funcionam em um mesmo período de tempo
- Funcional – mesma função/funcionalidade

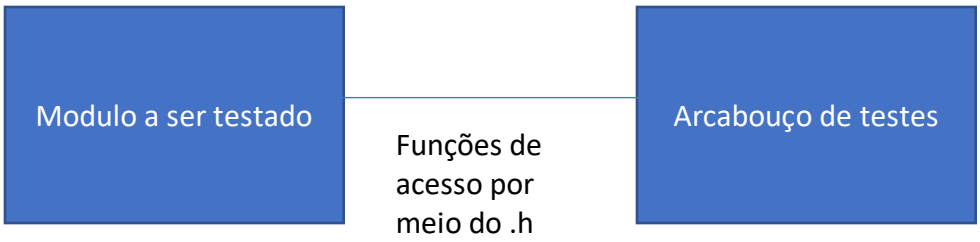
- Abstração de dados

- Uma união de conceitos (ex: TAD)

Teste automatizado

1) Objetivo
   Testar de forma automática um conjunto de casos de teste na forma de um script e gerando um log de saída com a analise entre o resultado esperado e o obtido.
   Obs: A partir do primeiro retorno esperado diferente do obtido no log de saída, todos os resultados de execução de casos de teste não são confiáveis.

2) Framework de teste



   - Genérica: lê o script e gera o log
   - Específica: Traduz o script para as funções de acesso do modulo
   - Parte genérica do arcabouço esta no Arcaboucoteste.lib, e a especifica esta no testarv.c. O conector entre os dois é a função efetuarComando, chamada na parte genérica. Aonde a entrada é o comando e a saída o retorno da função testada.

3) Script de teste
   - //-> Comentário
   - == Caso de teste -> testa determinada situação
   - =comando de teste -> associada a uma função de acesso
     Obs: teste completo -> casos de teste para todas as condições de retorno de cada função de acesso do modulo. (exceto falta de memoria)

4) Log de saída
    - ==caso1 – esperado coincide com obtido
    - ==caso2 – erros não esperados 1>> função esperava 0 e retorno 1
    - Erros esperados: 1>> 2>>
      Obs: Com a função recuperar é possível zerar o contador de erros, essa função server como um debug.

5) Parte especifica
    - A parte especifica que necessita ser implementada para que o framework(arcabouço) possa acoplar na aplicação chama-se hotspot.
      Ex: testearv.c

Processo de desenvolvimento em engenharia de software

Demanda (cliente) -> analista de negócios (contrato) ->Líder de projeto

- Projeto {tamanho (Ponto de função) ,esforço,recursos,prazo}
- Estimativa (pode ser feita por ponto de função, aonde é cobrado por cada item desenvolvido)
- Planejamento
- Acompanhamento

A. Requisitos:
    - Elicitaçao (coletar as informações do cliente)
    - Documentação
    - Verificação
    - Validação
B. Analise e projeto
    - Projeto lógico: modelagem de dados em UML
    - Projeto físico: tabelas do banco de dados
C. Implementação
    - Programas
    - Teste unitário (caso esteja tudo ok, passe para próxima fase)
D. Testes

- Teste integrado (Build)

E. Homologação (beta)
    - Sugestões
    - Erros
F. Implantação
    - Gerencia de configurações (Baseline)
    - Qualidade de software (mesura etapas)