Anotações de Programação Modular

Professor: Flávio
Turma: 3WB
Aluno: **Guilherme Dantas de Oliveira**

Aula 1

Bibliografia: “Programação Modular” de Arndt Von Staa OU O CADERNO DE ANOTAÇÕES
Os trabalhos estão organizados em:

- **T1: Arcabouço do Teste**
- **T2 e T3: Trabalho do Período**
- **T4: Instrumentação**

As notas dos trabalhos consistirá em 50% da nota do trabalho em si e 50% das anotações feita pelo aluno em sala de aula, em formato DIGITAL.
As provas P1 e P2 são com consulta, com reposição uma semana depois de cada aplicação. Em caso de a reposição também coincidir com alguma prova de outro curso, agendar com o professor outro horário e data para repor a prova. As datas das provas são 10/10 e 10/12.
Os graus são calculados pela seguinte média ponderada:
G1 = (T1 + 2*T2 + 2*P1)/5
G2 = (T3 + 2*T3 + 2*P2)/5

★ **Vantagens de Programação Modular:**

- Vencer barreiras de complexidade
- Facilita o trabalho em grupo (paralelismo)
- Reúso de componentes
- Facilita a criação de um acervo ou coleção
- Desenvolvimento incremental: estratégia de desenvolvimento que secciona o projeto em partes a serem implementadas. Neste caso, os módulos.
- Aprimoramento individual de um módulo, sem precisar compilar o projeto todo.
- Facilita a administração de baselines: versões estáveis da build e as versões de cada módulo do projeto que funcionam seguramente.

Aula 2

★ Módulo

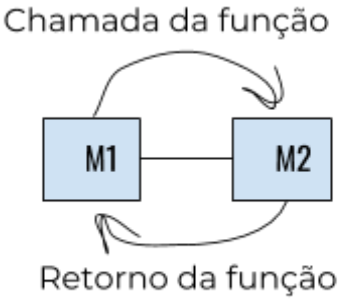
- Definição física: unidade de compilação independente (como um .c)
- Definição lógica: trata de um único conceito

★ Abstração do Sistema

- Abstrair é selecionar o que é necessário e o que não é necessário para o sistema. É o passo em que se define o **escopo**.
- Níveis de abstração: SISTEMA - PROGRAMAS - MÓDULOS - FUNÇÕES - BLOCO DE CÓDIGO - LINHAS DE CÓDIGO
- **Artefatos** são itens com identidade própria criados dentro um processo de desenvolvimento. O artefato pode ser versionado (ou seja, que participa de um controle de versão). Um arquivo é um exemplo de artefato.
- Construto ou **build**: resultado apresentável, mesmo que incompleto, do sistema.

★ Interface

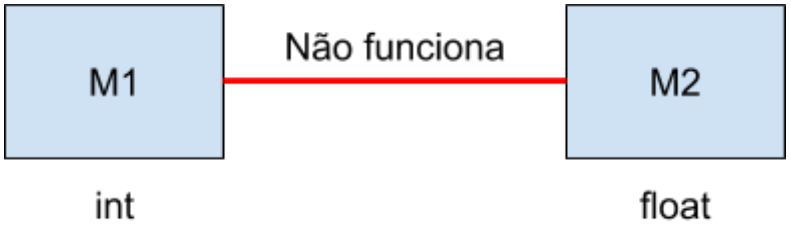
- Definição: mecanismo de troca de dados, estados e eventos entre elementos de um mesmo nível de abstração.
- Entre sistemas: arquivos
- Entre módulos: métodos ou funções de acesso
- Entre funções: passagens de parâmetro
- Entre blocos de código: variável global
- **Relacionamento Cliente-Servidor:** o módulo M1 chama a função do módulo M2. Portanto, M1 é o cliente e M2 é o servidor desta relação.



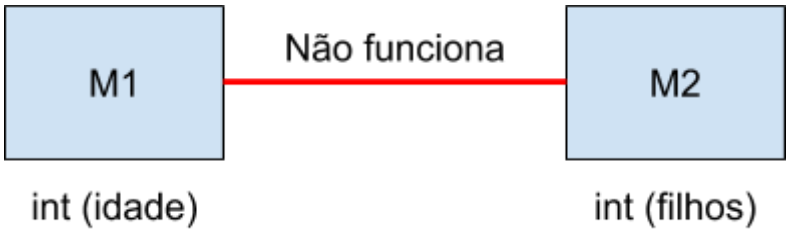
- **Caso especial**: o **callback** é quando o cliente faz uma chamada de função e o servidor requer mais dados do cliente para retornar a função. Nesse momento, há uma inversão de papéis entre os módulos.
- **Interface oferecida por terceiros**: ambos M1 e M2 usam o mesmo tipo de struct “aluno”. Para evitar futuras adversidades da duplicidade de dados, vincula-se a estes módulos um arquivo header para padronizar definições de structs, funções e variáveis destes módulos.

Aula 3

- **Interface em detalhe:**
  - Sintaxe: regras para uma troca de dados do mesmo tipo. (uma função não pode receber valores de tipos distintos, como int e float)



  - Semântica: do mesmo domínio de valores (significado). (O retorno de uma função deve estar bem definido no seu sentido. O que retorna?)

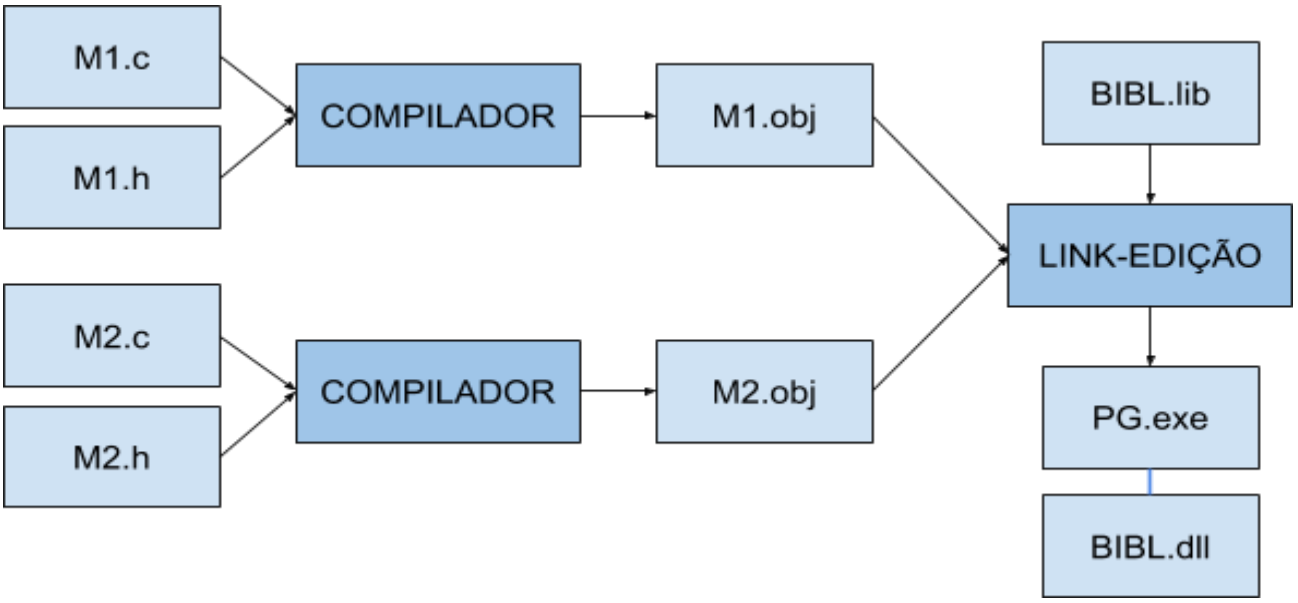


- **Análise de Interfaces:**

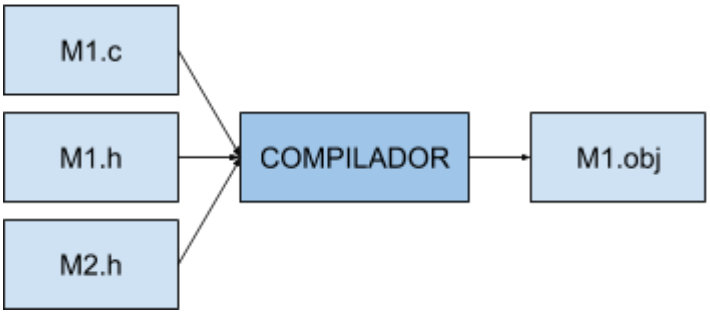
```
tpDadosAluno *obterDadosAluno(int mat);
```

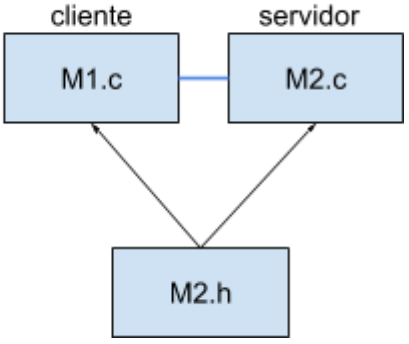
  - Interface esperada pelo cliente: um ponteiro para dados do aluno correto ou NULL.
  - Interface esperada pelo servidor: um inteiro válido representando a matrícula do aluno.
  - Interface esperada por ambos: a struct tpDadosAluno

★ **Processo de Desenvolvimento:** o processo por trás da geração de um constructo se dá através do seguinte fluxograma…



Na compilação de um dado módulo, todas as interfaces que tal módulo emprega serão “convocadas” na etapa em que se compilam os módulos, resultando num único objeto. Veja abaixo a ilustração deste processo:





Cada item deste fluxograma tem um nome particular:

.c	Módulo de Implementação
.h	Módulo de Definição
.lib	Biblioteca Estática
.dll	Biblioteca Dinâmica

★ **Bibliotecas estáticas e dinâmicas:** as vantagens e desvantagens de cada biblioteca está expressa na seguinte tabela…

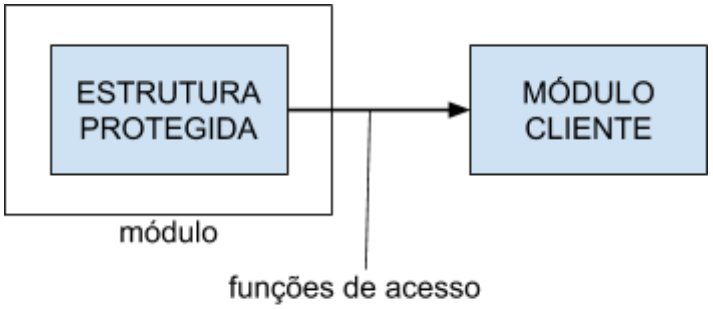
	Estática (.lib)	Dinâmica (.dll)
VANTAGEM	A .lib em tempos de link-edição já é acoplada à aplicação executada.	Só é carregada uma instância de biblioteca dinâmica mesmo que várias aplicações a acessem.
DESVANTAGEM	Existe uma cópia desta biblioteca estática para cada executável armazenada que a utiliza.	A .dll precisa estar na máquina para a aplicação funcionar.

★ **Módulo de definição (.h):**
- Interface do módulo
- Contém protótipos das funções de acesso
- Interface fornecida por terceiros
- Documentação voltada para o programador do módulo cliente

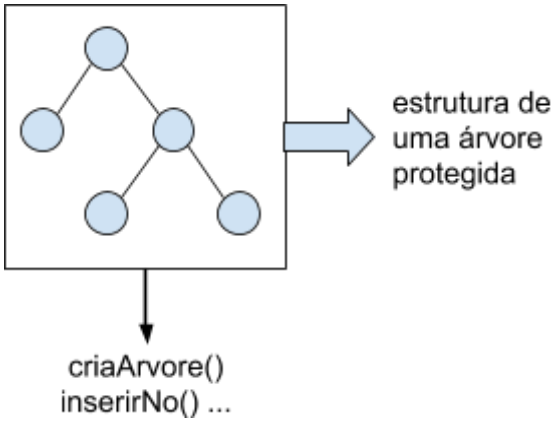
★ **Módulo de implementação (.c):**
- Código das funções de acesso
- Códigos e protótipos das funções internas
- Variáveis internas ao módulo
- Documentação voltada para o programador do módulo servidor

Aula 4

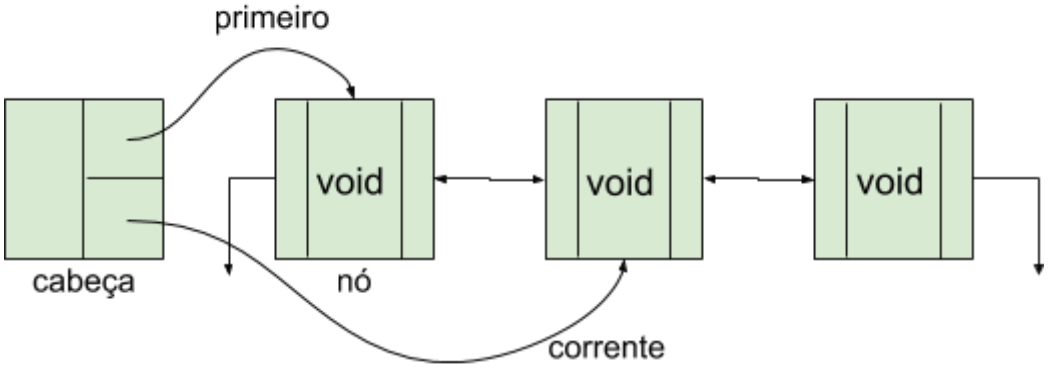
★ **Tipo abstrato de dados (TAD):** é uma estrutura encapsulada em um módulo que somente é conhecido pelos módulos clientes através das funções de acesso disponibilizada na interface.



No nosso caso, a estrutura protegida se trata de um TAD de Árvore, e as funções de acesso manipulam estes dados internos…



❑ **Exemplo:** Num TAD de Lista há duas structs: Nó e Cabeça. Tudo que fizermos dentro do módulo será através das funções de acesso.



Para administrar um TAD Lista, usamos funções de acesso como:

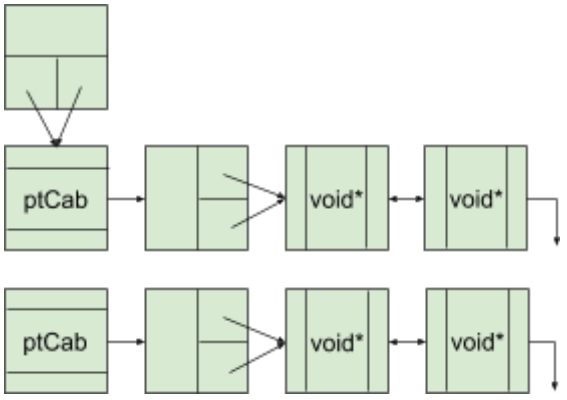
```
ir_prox(ptCab plista)
ir_ant(ptCab plista)
obter(ptCab plista)
criarLista(ptCab *plista) (1)
inserirNo(ptCab plista, void *conteudo) (2)
```

Funções como a (1) empregam a chamada **passagem de parâmetro por referência**: o endereço de uma variável no módulo cliente que terá seu valor alterado dentro da função de acesso. No caso desta função, o parâmetro é o endereço da struct Cabeça (apontará para a cabeça que terá posse da lista criada).

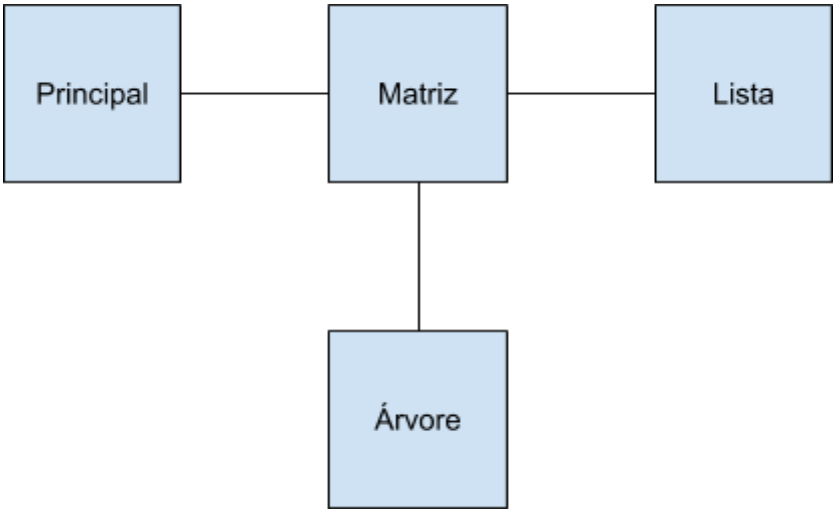
Já funções como a (2) empregam a chamada **passagem por valor**: acessamos o valor da variável para utilizá-la internamente, sem alterar o seu valor. No caso, o struct tpCab* é simplificado em ptCab (através de #typedef). Dentro do módulo servidor, há acesso permitido para mexer na estrutura protegida.

Assim, podemos montar uma matriz 2x2 por meio do TAD de Lista:

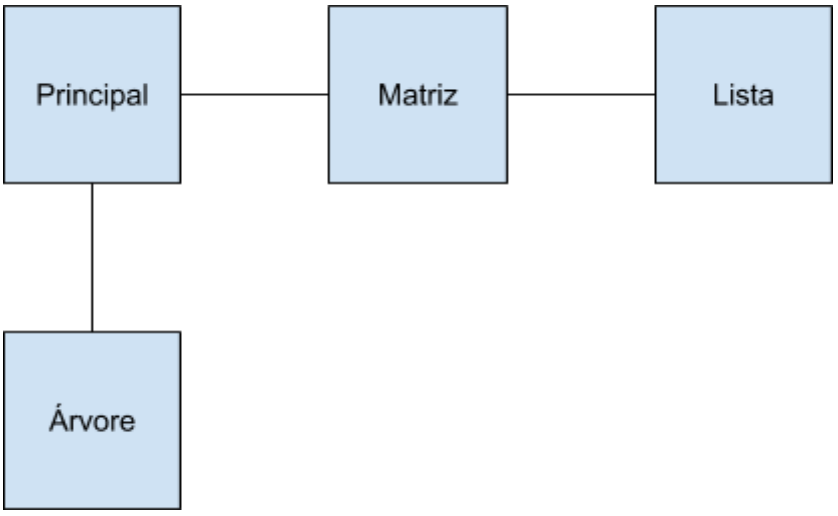
```
criaLista(p1);
criaLista(p2);
inserirNo(p2,NULL);
inserirNo(p2,NULL);
inserirNo(p1,p2);
criaLista(p3);
inserirNo(p3,NULL);
inserirNo(p3,NULL);
inserirNo(p1,p3);
```



Deve-se ressaltar que, a partir do momento em que reciclamos o código de Lista na implementação do módulo Matriz, o módulo principal não precisa fazer chamadas de funções de acesso do módulo Lista, pois o módulo Matriz internamente faz tais chamadas.



No grafo ao lado, podemos concluir que o módulo Matriz genérico necessita de Lista para sua estrutura, e que pode armazenar árvores em cada elemento. Contudo, este modelo impossibilita que o módulo principal crie árvores fora de uma matriz.



Já neste modelo, podemos perceber que é possível criar Árvores e Matrizes individualmente, mas também Matrizes de Árvores. A implementação de Matrizes cujos elementos são árvores cabe ao módulo Principal, contudo.

★ **Encapsulamento**: propriedade relacionada com a proteção dos elementos que compõem o módulo.

- Objetivo: facilitar a manutenção e impedir a utilização indevida da estrutura do módulo.
- Outros tipos de encapsulamento: de documentação
  - Doc. interna: .c (módulo de implementação)
  - Doc. externa: .h (módulo de definição)
  - Doc. de uso: manual de usuário
- De código - blocos de código visíveis apenas:
  - Dentro de módulo
  - Dentro de outro bloco de código (por exemplo: conjunto de comandos dentro de um loop)
  - Código de uma função
- De variável:
  - Private: encapsulamento no objeto
  - Public: global na Programação Orientada a Objetos
  - Global: global fora de Programação Orientada a Objetos
  - Global Static: no módulo
  - Protected: estrutura de herança (classes filhas herdam)
  - Static: global à classe (Programação Orientada a Objetos)

Aula 5

★ **Acoplamento:** propriedade relacionada com a interface entre os módulos.

- Conector - item de einterface. Ex: funções de acesso, variável global.

A qualidade de uma interface é medida pela:

- Quantidade de conectores - tudo na interface é necessário? É suficiente para que o código funcione?
- Tamanho do conector - quantidade de parâmetros de uma função, que pode ser minimizada ao agruparmos as variáveis em dados estruturados (struct de Endereço, de Data etc).
- Complexidade do conector - explicar na documentação, utilizar mnemônicos.

OBS: um bom acoplamento é o desejável; um alto acoplamento é quando as funções de acesso são altamente entranhadas com parâmetros.

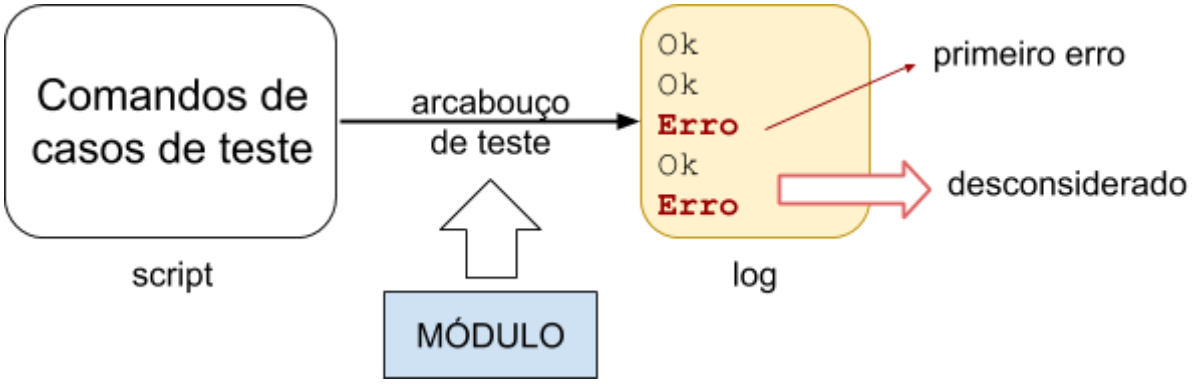
★ **Coesão:** propriedade relacionada com o grau de interligação dos elementos que compõem um módulo. Está diretamente ligada com a integridade do conceito de um módulo. Os níveis de coesão são…

- Incidental - pior coesão é quando dois conceitos que não tem qualquer semelhança estão no mesmo módulo.
- Lógica - elementos logicamente relacionados com uma ideia por trás (processa, calcula…).
- Temporal - itens que funcionam num mesmo período de tempo ou etapa (de_para, cancelar_matrícula…)
- Procedural: itens que são acionados em sequência / ordem lógica (linhas de um arquivo Batch ou .bat).
- Funcional: funções semelhantes (GeraRelatórioReceita, GeraRelatórioQualidade…)
- Abstração de dados: trata de um único conceito (TAD)

Aula 6

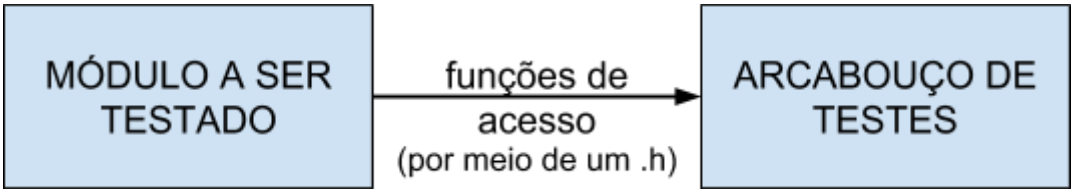
Teste Automatizado

★ **Objetivo:** é um script que faz o trabalho de testar de forma automática um módulo, recebendo um conjunto de casos teste e gerando um log de saída com a análise entre o resultado esperado e o objetivo.



- OBS: a partir do primeiro retorno esperado diferente do obtido no log de saída, todos os resultados de execução de casos de testes não são mais confiáveis. Isto é, podem haver falsos positivos e erros que dependem de erros.

★ **Framework de teste:** o arcabouço de teste possui uma parte genérica e uma parte específica



- Genérica: lê o script e gera o log
- Específica: traduz o script para funções de acesso do módulo a ser testado.
- No caso: a parte genérica está no Arcabouçoteste.lib e a específica está no TESTARV.C. O único conector entre estes dois é a função EfetuarComando, que é chamada pela parte genérica, onde a entrada é o comando e a saída é o retorno da função a ser testada.

★ **Script de Teste:** uma linguagem com os seguintes comandos…

- *//* é comentário.
- **==** é caso de teste: deseja testar um comando em um determinado contexto geral. Exemplo: excluir o nó raiz, excluir o nó intermediário, excluir nó final (folha).
- **=** é comando: chama uma função de acesso, sucedido de argumentos

- OBS: um teste completo é aquele que testa todas as condições de retorno de cada função de acesso do módulo (menos o estouro de memória).

★ **Log de Saída:** uma linguagem com as seguintes mensagens…
- ESPERADO COINCIDE COM OBTIDO: ==caso1
- ERROS NÃO ESPERADOS: 1 >> Função esperava 0 e retorno 1
- ERROS ESPERADOS: 1>> 0>>
  - Graças à função recuperar, zeramos o contador de erros caso quisermos testar o arcabouço de teste, não contabilizando como um erro não-previsto.

★ **Parte Específica:** A parte específica que necessita ser implementada para que o framework (arcabouço) possa acoplar na aplicação chama-se hotspot. Ex: TESTARV.C. A única função deste módulo é EfetuarComando, que recebe um comando de teste e executa com módulos a serem testados, retornando se o valor obtido coincide com o esperado ou se a estrutura do script está errada (para parte genérica).

Aula 7

Processo de Desenvolvimento em Engenharia de Software

- ★ A **demanda** vem do **cliente** para a empresa na figura do **analista de negócios**, que trata dos contratos, e define um **líder de projeto.**
    - O Líder de projeto cria o projeto, isto é, o **esforço**, os **recursos**, o tamanho (ponto de função) e o **prazo**. Para isso, ele dá uma estimativa (**deadline**). É ele quem vai planejar o projeto e supervisionar / acompanhar o seu desenvolvimento no decorrer do processo.
- ★ O processo é delineado pelas seguintes etapas:

A. Requisitos
   a. Elicitação
   b. Documentação
   c. Verificação
   d. Validação

B. Análise e projeto
   a. Projeto lógico: modelagem de dados em M.E-R, UML...
   b. Projeto físico: entidades como tabelas, módulos, arquivos...

C. Implementação
   a. Programas
   b. Teste Unitário: caso não apresente nenhum erro, passa para a próxima fase.

D. Testes
   a. Tese Integrado - caso haja erros nesta etapa, o projeto retrocede ao item C.

E. Homologação: o cliente faz uso da aplicação e julga erros ou modificações (feedback), que será considerada no retrabalho (custeado).
   a. Erros
   b. Sugestões

F. Implantação
   a. Gerência de configuração
   b. Qualidade de software