```
/**
    Curso de C++ ModernoPatricia Sette Câmara Haizer

    Lista de exercícios Módulo 03: Objetos: armazenamento, inicialia

    @author Arthur Nunes de Paiva Santos Queiroz
    @version 03/2017
*/

*) Respostas dos exercícios do slide (p. 62)

1) Definindo-se "secret" da seguinte forma:

        void secret(Id* id){
                delete id;
        }

Causa "segmentation fault" nos dois primeiros casos (ex. ponteiro, ex.
vetor) e "abort" nos dois últimos (variável automática, variável
estática).

2) Em minha plataforma (gcc version 4.8.4 (Ubuntu
4.8.4-2ubuntu1~14.04.3), o objeto Entropy é construido antes de RandNum.

3) Eles são destruídos na ordem reversa em que foram declarados. Caso
não fosse assim, não seria possível garantir que
os destrutores de objetos da pilha que dependem em objetos declarados
anteriormente seriam executados com dependências válidas.
Fonte: http://stackoverflow.com/a/1245865/702828

4) Este exercício não foi realizado.

5)      Os operadores "new" e "delete" utilizam "malloc" e "free".
https://gcc.gnu.org/viewcvs/gcc/trunk/libstdc%2B%2B-v3/libsupc%2B%2B/
new_op.cc?view=markup
https://gcc.gnu.org/viewcvs/gcc/trunk/libstdc%2B%2B-v3/libsupc%2B%2B/
del_op.cc?view=markup

6) "Value v();" representa um caso de "vexing parse" em C++. A
expressão aparentemente possui duas interpretações:
- A definição de uma varíavel "v" do tipo Value;
- A declaração de uma função "v", sem parâmetros, que retorna um objeto
do tipo Value.
O padrão C++ define a expressão em questão sempre deve ser interpretada
da segunda forma.
Fontes:         http://stackoverflow.com/q/1424510/702828
                        https://en.wikipedia.org/wiki/Most_vexing_parse

7)
PersonPOD a;                    // Variáveis inicializadas com lixo.
PersonPOD b();                  // Compilador interpreta como
declaração de função e acusa erro.
PersonPOD c{};                  // Variáveis initializadas com zero.
Person d;                               // Compilador reclama que não
```

```
há construtor default para a classe.
Person e("eu");                 // Inicializa objeto corretamente.
PersonPOD f("eu");              // Compilador reclama que nenhum
construtor padrão foi encontrado.
Person g{"eu"};                 // Inicializa objeto corretamente.
PersonPOD h{"eu"};              // Inicializa objeto corretamente.
Person i("eu", 30);             // Inicializa objeto corretamente.
Person j{"eu", 30};             // Inicializa objeto corretamente.
PersonPOD k("eu", 30);  // Compilador reclama que não há construtores
viáveis.
PersonPOd l{"eu", 30};  // Objeto é inicializado corretamente.


================================================================
*) Referências
```