Lista de Exercícios 06
Escopo

Exercício 01) Considere a classe `Reuse` a seguir implementada em Java.

```
01: class Reuse {
02:   Reuse Reuse(Reuse Reuse) {
03:     Reuse: for (;;) {
04:       if (Reuse.Reuse(Reuse) == Reuse)
05:         break Reuse;
06:     }
07:     return Reuse;
08:   }
09: }
```

Com base nisso, responda.

a) Mostre todos os contextos nos quais a palavra `Reuse` está sendo usada em uma definição. Por exemplo, na linha 03 está se definindo um rótulo de nome `Reuse`.
b) Agora, associe cada uso da palavra `Reuse` com sua definição encontrada no item `(a)`. Por exemplo, o uso de `Reuse` da linha 05 está fazendo referência à definição da linha 03. Nesse caso, se o comando da linha `05` for executado, então o fluxo do programa será desviado para a linha `07`.

Exercício 02) Considere o programa a seguir escrito em JavaScript.

```
01: var x = 1;
02: if (true) {
03:   var x = 2;
04:   if (true) {
05:     var x = 3;
06:     console.log(x);
07:   }
08:   console.log(x);
09: }
10: console.log(x);
```

scope.js

```
$ node scope.js
3
3
3
```

Ao executar esse programa pelo `node` observa-se que o valor `3` é impresso três vezes. Baseado nisso, o que se pode dizer sobre o escopo de variáveis declaradas como `var` em JavaScript? E se trocar `var` por `let`, o que acontece?

Exercício 03) Considere o programa a seguir escrito em Bash.

```
01: #!/bin/bash
02: x=1
03: function g() { echo $x; x=2; }
04: function f() { local x=3; g; echo $x; }
05: f;
06: echo $x;
```

scope.sh

```
$ chmod +x scope.sh
$ ./scope.sh
3
2
1
```

Ao executar esse programa pela *shell* observa-se que primeiro é impresso o valor `3`, depois `2` e então `1`. Baseado nisso, o que se pode dizer sobre o escopo de variáveis declaradas com `local` em Bash? E se remover `local`, o que acontece?

Exercício 04) Considere o seguinte trecho de código em ADA.

```
01: procedure A is
02:   u : INTEGER;
03:   procedure B is
04:     v : INTEGER;
05:     procedure C is
06:       x : INTEGER;
07:       procedure D is
08:         u : INTEGER;
09:       begin
10:         null;
11:       end D;
12:       procedure E is
13:         v: INTEGER;
14:       begin
15:           u := 7;
16:       end E;
17:     begin
18:       null;
19:     end C;
20:     procedure F is
21:       y : INTEGER;
22:       procedure G is
23:         x : INTEGER;
24:       begin
25:         null;
26:       end G;
27:     begin
28:       u := 10;
29:     end F;
30:   begin
31:     null;
32:   end B;
33: begin
34:   null;
35: end A;
```

a) Identifique quais variáveis e subprogramas são visíveis em cada um dos subprogramas desse trecho de código.
b) Agora suponha que novos requisitos demandem que a variável `u` de `D` possa ser acessada por `G`. Quais modificações seriam necessárias no programa? Cite erros que poderiam ocorrer em situações como essa.

Exercício 05) Considere o seguinte programa com sintaxe similar a C.

```
01: void main() {
02:   int a, b, c;
03:   // ...
04: }
05: void fun1(void) {
06:   int b, c, d;
07:   // ...
08: }
09: void fun2(void) {
10:   int c, d, e;
11:   // ...
12: }
13: void fun3(void) {
14:   int d, e, f;
15:   // ...
16: }
```

Dadas as seguintes sequências de chamadas, quais variáveis estarão visíveis durante a execução da última função chamada no **escopo estático** e para o **escopo dinâmico**? Inclua o escopo de declaração (nome da função) de cada uma das variáveis que estiverem visíveis.

a) `main` chama `fun1`; `fun1` chama `fun2`; `fun2` chama `fun3`.
b) `main` chama `fun1`; `fun1` chama `fun3`.
c) `main` chama `fun2`; `fun2` chama `fun3`; `fun3` chama `fun1`.
d) `main` chama `fun3`; `fun3` chama `fun1`.
e) `main` chama `fun1`; `fun1` chama `fun3`; `fun3` chama `fun2`.
f) `main` chama `fun3`; `fun3` chama `fun2`; `fun2` chama `fun1`.