Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3641 – Lenguajes de Programación I
Septiembre–Diciembre 2025

Estudiante: Luis Isea
Carnet: 19-10175

Examen 1

(25 puntos)

14 de octubre de 2025

Índice

Pregunta 1

(a) Breve descripción del lenguaje escogido

Como mi nombre es Luis, la inicial del lenguaje debe ser una L, por lo que he decidido optar por Lua.

Lua es un lenguaje de programación open source multiparadigma, dinámicamente tipado creado en 1993 en la Pontificia Universidad Católica de Río de Janeiro (PUC-Rio), Brasil. Su nombre significa "Luna" en portugués. Fue diseñado desde sus inicios para ser embedible (integrado dentro de otros lenguajes o archivos), portable, pequeño, rápido y con una sintaxis simple.

Lua es implementado como una librería escrita en 'clean C' (usando el subset de instrucciones comunes de C estándar y C++) y corre interpretando bytecode con una máquina virtual basada en registros.

I. Alcances y asociaciones

Lua posee un sistema de **alcance léxico** (o estático). Esto significa que el ámbito de una variable está determinado por su posición en el código fuente (su estructura léxica), no por el momento de su ejecución.

Ventajas de la decisión de diseño

- **Predecibilidad y Seguridad:** El comportamiento del código es más fácil de razonar al saber exactamente con qué variables se trabaja.
- **Flexibilidad con closures:** A pesar que el alcance estático no es tan flexible como el dinámico, Lua permite usar los denominados closures (cierres) que son funciones definidas dentro de otras funciones y estas funciones internas (las anidadas dentro de las otras) pueden acceder a las vasriables de las funciones externas aunque teróricamente no forman parte de su alcance.

Desventajas de la decisión de diseño

- **Concepto Abstracto:** Los closures pueden ser difíciles de entender inicialmente, ya que no son muy comunes en otros lenguajes.
- **Gestión de Memoria:** Los closures mantienen referencias a variables que podrían haberse liberado, lo que requiere cuidado en el diseño.
- **Pérdida de Flexibilidad:** No permite alcance dinámico de forma nativa, aunque se puede emular con librerias externas o a través de la manipulación de los entornos al crearlos, lo que complica y alarga el programa.

II. Tipo de módulos y formas de importar y exportar nombres

Lua no tiene un sistema de módulos integrado en el núcleo del lenguaje, sino que utiliza un mecanismo simple y efectivo basado en su función `require`. Básicamente, se importa con require de un archivo lo que se exporte con return en este.

Exportar un módulo

```lua
-- archivo: mi_modulo.lua
local M = {} -- Crea una tabla vacía para el módulo

function M.saludar(nombre)
    print("Hola, " .. nombre)
end

M.constante_pi = 3.14159

return M
```

Listing 1: Exportar un módulo en Lua

Importar un módulo

```lua
-- archivo: main.lua
local miModulo = require("mi_modulo") -- No necesita la extensión .lua

miModulo.saludar("Maria") -- Output: Hola, Maria
print(miModulo.constante_pi) -- Output: 3.14159
```

Listing 2: Importar un módulo en Lua

III. Possibilidad de crear aliases, sobrecarga y polimorfismo

Aliases

Sí, Lua permite crear alias de forma directa:

```lua
local miFuncionLarga = unaTabla.unaFuncionMuyLarga
miFuncionLarga() -- Equivale a llamar a unaTabla.unaFuncionMuyLarga
```

Listing 3: Creación de alias en Lua

Sobrecarga

La **sobrecarga de funciones** (function overloading) se refiere a la capacidad de tener múltiples funciones con el mismo nombre pero diferentes parámetros (en tipo o número), donde el lenguaje selecciona automáticamente la implementación correcta basándose en los argumentos.

Lua no soporta sobrecarga nativamente, ya que cada definición de función sobrescribe la anterior.

Polimorfismo

Lua sí permite el polimorfismo. Tal como se apreciará en el ejemplo:

```lua
local Pajaro = {}
function Pajaro:volar()
    print("El pajaro aletea y vuela")
end

```

```lua
local Avion = {}
function Avion:volar()
    print("El avion enciende motores y despega")
end

-- Funcion polimorfica: acepta cualquier objeto que tenga metodo volar
function hacerVolar(objetoVolador)
    objetoVolador:volar()
end

hacerVolar(Pajaro)   -- Output: El pajaro aletea y vuela
hacerVolar(Avion)    -- Output: El avion enciende motores y despega
```

Listing 4: Polimorfismo en Lua

En Lua, cualquier tabla que tenga un método `volar()` puede ser pasada a `hacerVolar()`, demostrando polimorfismo.

IV. Herramientas para desarrolladores

Herramientas principales (Proporcionadas por Lua.org)

- **Intérprete:** El binario `lua` estándar - `https://www.lua.org/`
- **Compilador Bytecode:** `luac` para compilar a bytecode - Incluido en la distribución oficial
- **Debugger básico:** Librería `debug` integrada en el lenguaje

Herramientas externas

- **IDEs y Debuggers:**
  - **ZeroBrane Studio:** IDE completo con debugger visual - `https://studio.zerobrane.com/`
  - **Visual Studio Code:** Con extensión Lua - `https://marketplace.visualstudio.com/items?itemName=sumneko.lua`
- **Profilers:**
  - **LuaJIT Profiler:** Herramienta de profiling para LuaJIT - `https://luajit.org/ext_profiler.html`

Frameworks y entornos (Externos)

- **LÖVE2D:** Framework para creación de juegos 2D - `https://love2d.org/`
- **Solar2D:** Motor de juegos en Lua (fork de Corona SDK) - `https://solar2d.com/`
- **OpenResty:** Plataforma web basada en Nginx y Lua - `https://openresty.org/`

Gestión de paquetes

- **LuaRocks:** Gestor de paquetes oficial para Lua - `https://luarocks.org/`

(b) Implemente los siguientes programas en el lenguaje escogido

Las programas fueron implementados y se subieron al siguiente repositorio de GitHub.

Repositorio GitHub

`https://github.com/lmisea/lenguajes-prog.git`

Pregunta 2

Considere el siguiente programa escrito en pseudo-código con los valores $X = 1$, $Y = 7$, $Z = 5$:

```
int a = Y + Z + 1, b = X + Y + 1, c = Z + Y + 1;
sub R (int b) {
    a := b + c - 1
}
sub Q (int a, sub r) {
    b := a + 1
    r(c)
}
sub P(int a, sub s, sub t) {
    sub R(int a) {
        b := c + a + 1
    }
    sub Q (int b, sub r) {
        c := a + b
        r(c + a)
        t(c + b)
    }
    int c := a + b
    if (a < 2 * (Y + Z + 1)) {
        P(a + 2 * (Y + Z + 1), s, R)
    } else {
        int a := c + 1
        s(c * a, R)
        Q(c * b, t)
    }
    print(a, b, c)
}
P(a, Q, R);
print(a, b, c)
```

(a) Alcance estático y asociación profunda

Valores iniciales:

- $a = Y + Z + 1 = 7 + 5 + 1 = 13$
- $b = X + Y + 1 = 1 + 7 + 1 = 9$
- $c = Z + Y + 1 = 5 + 7 + 1 = 13$

Análisis de la ejecución:

1. **Llamada inicial:** P(13, Q, R)
2. **Dentro de P:**

- Se define R local: $b := c + a + 1$
- Se define Q local: $c := a + b$, $r(c + a)$, $t(c + b)$
- $c := a + b = 13 + 9 = 22$
- $a = 13 < 2 \times (7 + 5 + 1) = 26 \rightarrow$ verdadero
- **Llamada recursiva:** P(13 + 26 = 39, Q, R-local)

3. **Segunda llamada a P(39, Q, R-local):**
   - $c := a + b = 39 + 9 = 48$
   - $a = 39 < 26 \rightarrow$ falso
   - $a := c + 1 = 48 + 1 = 49$
   - **Llamada:** s(c * a, R) = Q(48 * 49 = 2352, R-local)
4. **Dentro de Q global con R-local:**
   - $b := a + 1 = 2352 + 1 = 2353$
   - $r(c) = R - local(48)$
   - **Dentro de R-local:** $b := c + a + 1 = 48 + 39 + 1 = 88$
5. **Continuación en P:** Q(48 * 9 = 432, R)
   - **Dentro de Q local:** $c := a + b = 39 + 432 = 471$
   - $r(c + a) = R(471 + 39 = 510)$
   - **Dentro de R global:** $a := b + c - 1 = 510 + 13 - 1 = 522$
   - $t(c + b) = R(471 + 432 = 903)$
   - **Dentro de R global:** $a := b + c - 1 = 903 + 13 - 1 = 915$
6. **Print en P:** print(39, 88, 471)
7. **Retorno a primera llamada:** print(13, 88, 471)
8. **Print final:** print(915, 9, 13)

Salida:

```
39, 88, 471
13, 88, 471
915, 9, 13
```

(b) Alcance dinámico y asociación profunda

Valores iniciales:

- $a = 13$, $b = 9$, $c = 13$

Análisis de la ejecución:

1. **Llamada inicial:** P(13, Q, R)

2. **Dentro de P:**
   - $c := a + b = 13 + 9 = 22$
   - $a = 13 < 26 \rightarrow$ verdadero
   - **Llamada recursiva:** P(39, Q, R-local)
3. **Segunda llamada a P(39, Q, R-local):**
   - $c := a + b = 39 + 9 = 48$
   - $a = 39 < 26 \rightarrow$ falso
   - $a := c + 1 = 48 + 1 = 49$
   - **Llamada:** Q(2352, R-local)
4. **Dentro de Q (global):**
   - $b := a + 1 = 2352 + 1 = 2353$
   - $r(c) = R - local(48)$
   - **Dentro de R-local:** $b := c + a + 1 = 48 + 49 + 1 = 98$
5. **Continuación en P:** Q(432, R)
   - **Dentro de Q (local):** $c := a + b = 49 + 432 = 481$
   - $r(c + a) = R(481 + 49 = 530)$
   - **Dentro de R (global):** $a := b + c - 1 = 530 + 48 - 1 = 577$
   - $t(c + b) = R(481 + 432 = 913)$
   - **Dentro de R (global):** $a := b + c - 1 = 913 + 48 - 1 = 960$
6. **Print en P:** print(49, 98, 481)
7. **Retorno a primera llamada:** print(13, 98, 481)
8. **Print final:** print(960, 9, 13)

Salida:

```
49, 98, 481
13, 98, 481
960, 9, 13
```

(c) Alcance estático y asociación superficial

Valores iniciales:

- $a = 13$, $b = 9$, $c = 13$

Análisis de la ejecución:

1. **Llamada inicial:** P(13, Q, R)
2. **Dentro de P:**

- $c := a + b = 13 + 9 = 22$
- $a = 13 < 26 \rightarrow$ verdadero
- **Llamada recursiva:** P(39, Q, R-local)

3. **Segunda llamada a P(39, Q, R-local):**
   - $c := a + b = 39 + 9 = 48$
   - $a = 39 < 26 \rightarrow$ falso
   - $a := c + 1 = 48 + 1 = 49$
   - **Llamada:** Q(2352, R-local)
4. **Dentro de Q (global):**
   - $b := a + 1 = 2352 + 1 = 2353$
   - $r(c) = R - local(48)$
   - **Dentro de R-local:** $b := c + a + 1 = 48 + 39 + 1 = 88$
5. **Continuación en P:** Q(432, R)
   - **Dentro de Q (local):** $c := a + b = 39 + 432 = 471$
   - $r(c + a) = R(471 + 39 = 510)$
   - **Dentro de R (global):** $a := b + c - 1 = 510 + 13 - 1 = 522$
   - $t(c + b) = R(471 + 432 = 903)$
   - **Dentro de R (global):** $a := b + c - 1 = 903 + 13 - 1 = 915$
6. **Print en P:** print(39, 88, 471)
7. **Retorno a primera llamada:** print(13, 88, 471)
8. **Print final:** print(915, 9, 13)

Salida:

```
39, 88, 471
13, 88, 471
915, 9, 13
```

(d) Alcance dinámico y asociación superficial

Valores iniciales:

- $a = 13$, $b = 9$, $c = 13$

Análisis de la ejecución:

1. **Llamada inicial:** P(13, Q, R)
2. **Dentro de P:**
   - $c := a + b = 13 + 9 = 22$

- $a = 13 < 26 \rightarrow$ verdadero
- **Llamada recursiva:** P(39, Q, R-local)

3. **Segunda llamada a P(39, Q, R-local):**
   - $c := a + b = 39 + 9 = 48$
   - $a = 39 < 26 \rightarrow$ falso
   - $a := c + 1 = 48 + 1 = 49$
   - **Llamada:** Q(2352, R-local)
4. **Dentro de Q (global):**
   - $b := a + 1 = 2352 + 1 = 2353$
   - $r(c) = R - local(48)$
   - **Dentro de R-local:** $b := c + a + 1 = 48 + 49 + 1 = 98$
5. **Continuación en P:** Q(432, R)
   - **Dentro de Q (local):** $c := a + b = 49 + 432 = 481$
   - $r(c + a) = R(481 + 49 = 530)$
   - **Dentro de R (global):** $a := b + c - 1 = 530 + 48 - 1 = 577$
   - $t(c + b) = R(481 + 432 = 913)$
   - **Dentro de R (global):** $a := b + c - 1 = 913 + 48 - 1 = 960$
6. **Print en P:** print(49, 98, 481)
7. **Retorno a primera llamada:** print(13, 98, 481)
8. **Print final:** print(960, 9, 13)

Salida:

```
49, 98, 481
13, 98, 481
960, 9, 13
```

Pregunta 3, 4 y 5

Las preguntas 3, 4 y 5 del examen fueron implementadas y subidas al siguiente repositorio de GitHub.

Repositorio GitHub

`https://github.com/lmisea/lenguajes-prog.git`