Paradigmas de la Programación – Examen Final

28 de Febrero de 2024

Apellido y Nombre: _____

1. **[10 pt.]** El siguiente texto explica lo que es un *thunk*.

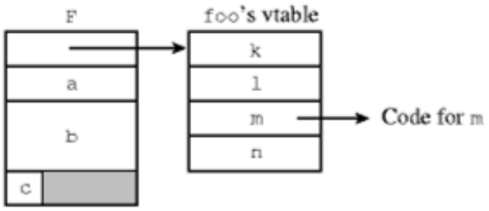
   > *A simple implementation of "call by name" might substitute the code of an argument expression for each appearance of the corresponding parameter in the subroutine, known as a "thunk".*

   Escriba el *thunk* resultante de compilar `f(array[i])` en el siguiente programa, donde los argumentos se pasan mediante la estrategia *call-by-name*:

```
1  int i;
2  int array[3] = { 0, 1, 2 };
3
4  i = 0;
5  f(array[i]);
6
7  int f(int j)
8  {
9      int k = j;
10     i = 2;
11     k = j;
12 }
```

2. **[10 pt.]** En esta imagen se observa cómo C++ guarda los métodos virtuales en una vtable. Explique cómo es el proceso mediante el cuál se usa la vtable en C++, explique la diferencia entre métodos virtuales y no virtuales en términos de *overhead* y *flexibilidad*, y qué decisión de diseño de C++ justifica que por defecto todos los métodos sean no virtuales.

```
class foo {
    int a;
    double b;
    char c;
public:
    virtual void k ( ...
    virtual int l ( ...
    virtual void m ();
    virtual double n( ...
    ...
} F;
```



3. Escriba en pseudocódigo un programa en el que:

   a) **[10 pt.]** una componente tenga efectos secundarios pero sea determinística

   b) **[10 pt.]** una componente no tenga efectos secundarios pero no sea determinística

4. **[20 pt.]** Cuáles son los valores de "y" y "z" al final del siguiente bloque, asumiendo que el pasaje de parámetros es a) por valor, b) por referencia y c) por valor-resultado?

```
{ int y;
  int z;
  y := 7;
  { int f(int x) {
      x := x+1;
      y := x;
      x := x+1;
      return y
    };
    int g(int x) {
      y := f(x)+1;
      x := f(y)+3;
      return x
    };
    z := g(y)
  }
}
```

5. **[10 pt.]** Dada la siguiente base de conocimiento, ¿qué va a contestar el intérprete si le preguntamos `digiriendo(rana,mosca).`, y cómo va a llegar a su respuesta?

```
digiriendo(X,Y) :- comio(X,Y).
digiriendo(X,Y) :-
        comio(X,Z),
        digiriendo(Z,Y).

comio(mosquito,sangre(juan)).
comio(rana,mosquito).
comio(gaviota,rana).
```

6. **[10 pt.]** Explique verbalmente qué va sucediendo en la ejecución del siguiente programa, ayudándose de diagramas de la pila de ejecución cuando lo considere necesario. No es necesario representar las variables locales, control links, access links, retorno de función ni ninguna otra información que no sea relevante al manejo de excepciones. Explique también qué imprime el programa.

```
class Ejemplo3 {
  public static void main(String args[]){
    try{
        System.out.println("primera sentencia del bloque
   try");
        int num=45/3;
        System.out.println(num);
    }
    catch(ArrayIndexOutOfBoundsException e){

   System.out.println("ArrayIndexOutOfBoundsException");
    }
    finally{
        System.out.println("bloque finally");
    }
    System.out.println("fuera del bloque
   try-catch-finally");
  }
}
```

7. **[10 pt.]** El siguiente texto explica los conceptos de *hot spot* y *frozen spot* en frameworks. También nos explica cómo funcionan los frameworks con orientación a objetos. Cuando instanciamos una aplicación con un framework basado en objetos, terminamos teniendo un sistema de objetos específico para esa aplicación, basado en el sistema de objetos provisto por el framework. Basándose en los conceptos de *concreto* y *abstracto* y *composición* y *subclases* que usa el texto, explique qué componentes de este sistema de objetos se podrían considerar *frozen spots* y cuáles se podrían considerar *hot spots*.

> *Software frameworks consist of frozen spots and hot spots. Frozen spots define the overall architecture of a software system, that is to say its basic components and the relationships between them. These remain unchanged (frozen) in any instantiation of the application framework. Hot spots represent those parts where the programmers using the framework add their own code to add the functionality specific to their own project.*
>
> *In an object-oriented environment, a framework consists of abstract and concrete classes. Instantiation of such a framework consists of composing and subclassing the existing classes.*

8. **[10 pt.]** El siguiente programa está escrito en Erlang. Encuentre en el texto del programa por lo menos dos características propias del paradigma de actores y relaciónelas con lo que conoce de Akka.

```
% Create a process and invoke the function web:start_server(Port, MaxConnections)
ServerProcess = spawn(web, start_server, [Port, MaxConnections]),

% Create a remote process and invoke the function
% web:start_server(Port, MaxConnections) on machine RemoteNode
RemoteProcess = spawn(RemoteNode, web, start_server, [Port, MaxConnections]),

% Send a message to ServerProcess (asynchronously). The message consists of a tuple
% with the atom "pause" and the number "10".
ServerProcess ! {pause, 10},

% Receive messages sent to this process
receive
        a_message -> do_something;
        {data, DataContent} -> handle(DataContent);
        {hello, Text} -> io:format("Got hello message: ~s", [Text]);
        {goodbye, Text} -> io:format("Got goodbye message: ~s", [Text])
end.
```

9. **[15 pt.]** Pueden elegir uno u otro de estos dos ejercicios de seguridad:

   A En el siguiente fragmento de código se está aplicando una estrategia de programación defensiva. Identifíquela **[10 pt.]** y describa cómo funciona **[5 pt.]**

```
1  def divide_numbers(dividend, divisor):
2      if divisor == 0:
3          raise ValueError("Cannot divide by zero.")
4      result = dividend / divisor
5      return result
6
7  def get_user_input():
8      try:
9          dividend = int(input("Enter the dividend: "))
10         divisor = int(input("Enter the divisor: "))
11         result = divide_numbers(dividend, divisor)
12         print(f"The result of the division is: {result}")
13     except ValueError as e:
14         print("Invalid input:", e)
15     except Exception as e:
16         print("An error occurred:", e)
17
18 get_user_input()
```

   B En el siguiente texto nos explican una característica muy interesante de Elm. Esta característica, ¿contribuye negativamente o positivamente a la seguridad del lenguaje? ¿por qué? ¿y a la robustez del lenguaje? ¿Qué mecanismos de programación defensiva podríamos aplicar en otro lenguaje para tener un comportamiento parecido al comportamiento que presenta Elm en este programa?

One of the guarantees of Elm is that you will not see runtime errors in practice. This is partly because Elm treats errors as data. Rather than crashing, we model the possibility of failure explicitly with custom types. For example, say you want to turn user input into an age. You might create a custom type like this:

```
type MaybeAge
  = Age Int
  | InvalidInput

toAge : String -> MaybeAge
toAge userInput =
  ...

-- toAge "24" == Age 24
-- toAge "99" == Age 99
-- toAge "ZZ" == InvalidInput
```

Instead of crashing on bad input, we say explicitly that the result may be an Age 24 or an InvalidInput. No matter what input we get, we always produce one of these two variants. From there, we use pattern matching which will ensure that both possibilities are accounted for. No crashing!

10. ¿Qué tipo de lenguaje de scripting observamos en el siguiente ejemplo? **[5 pt.]** Nombren por lo menos 2 características de este tipo de lenguajes de scripting **[10 pt.]**

```
1  // Player entity variables
2  entity player;
3  float playerSpeed = 100.0;
4
5  void main() {
6      // Initialize player entity
7      player = spawn();
8      setmodel(player, "player.mdl");
9      setsize(player, Vector(-16, -16, 0), Vector(16, 16, 72));
10     setorigin(player, Vector(0, 0, 0));
11     player.think = Player_Think;
12     player.movetype = MOVETYPE_WALK;
13     player.solid = SOLID_BBOX;
14
15     // Start the game loop
16     while (1) {
17         // Process keyboard input
18         if (key_down(K_LEFTARROW)) {
19             player.velocity_y = -playerSpeed;
20         } else if (key_down(K_RIGHTARROW)) {
21             player.velocity_y = playerSpeed;
22         } else {
23             player.velocity_y = 0;
24         }
```

```
25
26          // Update the player entity
27          setorigin(player, player.origin + player.velocity * frametime);
28
29          // Update the game state
30          progs_run();
31      }
32  }
```

Ejercicios para Libres

1. **[-10 pt.]** En el siguiente fragmento de texto, identifique porciones con semántica concurrente y explíquelas.

```
public class Counting {
    public static void main(String[] args) throws InterruptedException {
        class Counter {
            int counter = 0;
            public void increment() { counter++; }
            public int get() { return counter; }
        }

        final Counter counter = new Counter();

        class CountingThread extends Thread {
            public void  run() {
                for (int x = 0; x < 500000; x++) {
                    counter.increment();
                }
            }
        }

        CountingThread t1 = new CountingThread();
        CountingThread t2 = new CountingThread();
        t1.start(); t2.start();
        t1.join(); t2.join();
        System.out.println(counter.get());
    }
}
```

2. **[-10 pt.]** Qué imprime este programa, asumiendo que el pasaje de parámetros es a) por valor, b) por referencia y c) por valor-resultado?

```
z : integer;
procedure p (x:integer)
  x := x+1 ;
  z := z+2;

z := 1;
p(z);
write(z)
```