Paradigmas de la Programación – Recuperatorio del Segundo Parcial

19 de Junio de 2018

Apellido y Nombre: _____

1. **[20 pt.]** En Perl tenemos las palabras clave `my` y `local` para definir diferentes alcances de las variables, tal como se explica en el siguiente texto. Explique por qué el programa de ejemplo final imprime 1, después 2 y después 1 de vuelta.

   > `my` *marks a variable as private in a lexical scope, and* `local` *marks a variable as private in a dynamic scope.*
   > *It's easier to understand* `my`*, since that creates a local variable in the usual sense. There is a new variable created and it's accessible only within the enclosing lexical block, which is usually marked by curly braces.*

   ```
   sub Foo {
       my $x = shift;

       print "$x\n";
   }
   ```

   > *In that case,* `$x` *is private to the subroutine and its scope is enclosed by the curly braces. The thing to note, and this is the contrast to* `local`*, is that the scope of a* `my` *variable is defined with respect to your code as it is written in the file. It's a compile-time phenomenon.*
   > *To understand* `local`*, you need to think in terms of the calling stack of your program as it is running. When a variable is* `local`*, it is redefined from the point at which the local statement executes for everything below that on the stack, until you return back up the stack to the caller of the block containing the local.*

   Sabiendo esto, explique por qué el siguiente programa imprime 1, después 2 y después 1. Use diagramas de los diferentes estados por los que pasa la pila de ejecución.

   ```
   sub foo { print "$x\n"; }
   sub bar { local $x; $x = 2; foo(); }

   $x = 1;
   foo(); # prints '1'
   bar(); # prints '2'
   foo(); # prints '1'
   ```

2. **[10 pt.]** Dada la siguiente base de conocimiento en Prolog, liste los subobjetivos que hay que satisfacer para verificar si el predicado `not(obsesionado(mario)).` es cierto.

   ```
   ama(mario,maria).
   ama(juan,maria).
   ama(maria,juan).
   conservador(mario).
   obsesionado(X) :- ama(X,Y), not(ama(Y,X)), posesivo(X).
   posesivo(X) :- conservador(X).
   ```

3. **[20 pt.]** En el siguiente fragmento de C++, diga qué se imprime si se pasan los parámetros tal como están escritos y qué se imprime si modificamos la declaración con `p(int &y, int &z) {...}`. Explique por qué en componentes declarativas no hay diferencia entre pasaje de parámetros por valor y pasaje de parámetros por referencia.

```
#include < stdio.h >
int x=0;
void p(int,int);
void main(){
   int x = 1;
   p(x,x);
}
void p(int y, int z){
   x = x+1;
   y = y+1;
   z = z+1;
   printf("%d\n",x+y+z);
}
```

4. **[30 pt.]** Explique por qué el siguiente programa es un ejemplo de programación defensiva, identifique las líneas de código que implementan programación defensiva (10 pt.). Explique qué vulnerabilidad del lenguaje se está protegiendo mediante estas acciones, y cómo otros lenguajes evitan este tipo de vulnerabilidad. Por último, escriba en pseudocódigo cómo se podría tratar este problema con excepciones (20 pt.).

```
static void CreateRandomPermutation(int numbers[], int nNumbers)
{
    assert(numbers != NULL);
    assert(nNumbers >= 0);

    for (int i=0; i<nNumbers; i++)
        numbers[i] = i;

    for (int src=1; src<nNumbers; src++)
    {
        const int dst = GetRandomNumberIn(0, src);
        SwapNumbers(numbers, src, dst);
    }
}
```

5. **[10 pt.]** De los siguientes fragmentos de código, ¿cuál es declarativo? En el que no es declarativo ¿Qué fenómenos podemos encontrar que no son declarativos?

```
procedure FACTORIAL is
   N: integer;
   T: integer:= 1;
begin
   put("Input ");get(N);
   for K in 2..N loop
      T:= T*K;
   end loop;
   put("Factorial ");put(N);
   put("is ");put(T);newline;
end FACTORIAL;
```

```
factorial(0,1):-
        !.

factorial(N1,T2):-
        N2 is N1-1,
        factorial(N2,T1),
        T2 is N1*T1.
```

6. **[10 pt.]** En el siguiente fragmento, identifique en el código las porciones con semántica concurrente.

```
class Line
{
    public void getLine()
    {
        for (int i = 0; i < 3; i++)
        {
            System.out.println(i);
            try
            {
                Thread.sleep(400);
            }
            catch (Exception e)
            {
                System.out.println(e);
            }
        }
    }
}
class Train extends Thread
{
    Line line;
    Train(Line line)
    {
        this.line = line;
    }
    @Override
    public void run()
    {
        line.getLine();
    }
}
class GFG
{
    public static void main(String[] args)
    {
        Line obj = new Line();

        Train train1 = new Train(obj);
        Train train2 = new Train(obj);

        train1.start();
        train2.start();
    }
}
```