Examen – Lenguajes de Programación (CC41A)

Departamento de Ciencias de la Computación – Universidad de Chile

Profesor: Éric Tanter

25 de Junio 2007

sin apuntes

1. **(1.2pt)**

   a) Considere el siguiente programa Scheme:

   ```
   (define factor 3.14)

   (define (f x) (* factor x))

   (define (g x)
     (let ((factor 1.96))
       (f x)))
   ```

   ¿Es cierto que las funciones `f` y `g` siempre retornan el mismo resultado? ¿o siempre un resultado distinto? ¿por qué?

   b) En Common Lisp, una variable global introducida por `defvar` siempre tiene scope dinámico, y se le llama *variable especial*. Por convención, el nombre de una variable especial se prefija y postfija con `*`. Por ejemplo:

   ```
   (defvar *factor* 3.14)

   (defun (f x) (* *factor* x))

   (defun (g x)
     (let ((*factor* 1.96))
       (f x)))
   ```

   ¿Qué puede decir de las funciones `f` y `g` en este programa Common Lisp, y por qué?

   c) A su juicio, ¿Para qué pueden servir las variables especiales? Describa un ejemplo de aplicación posible.

2. **(1 pt)** El programa `yes` en un shell Unix como bash genera una infinidad de `y`:

   ```
   > yes
   y
   ```

```
y
y
...
```

Para componer programas en el shell, uno usa el operador pipe: `|`, por ejemplo:

```
> yes | head -2
y
y
```

¿Qué les indica esto sobre el régimen de evaluación del lenguaje de shell? ¿Qué lenguaje de programación visto en clase tiene la misma característica? Escriba el equivalente del programa `yes | head -2` en este lenguaje.

3. **(1 pt)** Considere el siguiente programa escrito en el lenguaje SuperClaro:

```
defun pre(x) { x = (cons 'a x);}
defun post(x) { x.addLast('d);}
a = ('b, 'c);
pre(a);
post(a);
print(a);
```

¿Cuál es la salida del programa? ¿`(b, c)` / `(a, b, c)` / `(b, c, d)` / `(a, b, c, d)`?
De ser `(a, b, c, d)` el resultado, ¿qué puede decir del lenguaje SuperClaro?
Explique sus respuestas.

4. **(1.2 pt)** Una de las novedades para el lenguaje Java es la introducción de cerraduras (closures). Por ejemplo, esto es la definición de una closure que toma dos enteros y los suma:

```
{int x, int y => x+y}
```

La declaración/inicialización de una variable local para esta closure es:

```
{int,int=>int} plus = {int x, int y => x+y};
```

   a) De el juicio de tipo correspondiente.

   b) ¿Qué hace el siguiente código? De el valor de `newpuntos`.

```
int extra = 3;
{int=>int} addExtra = {int x => x + extra};
List<int> puntos = {12, 34, 56};
List<int> newpuntos = Collections.map(puntos, addExtra);
```

   c) En Haskell uno puede implementar lo anterior de la siguiente manera:

```
extra = 3
addExtra = (\ x -> x + extra)
puntos = [12, 34, 56]
newpuntos = map puntos addExtra
```

   Relativo al sistema de tipos: ¿Por qué el código Haskell no incluye declaración de tipos? ¿Es menos "seguro" que el código Java? ¿Por qué?

5. **(1,6 pt)** Considerando los siguientes criterios:

   - estatus de las funciones/procedimientos: 1er orden, orden superior, 1a clase
   - scope de variables: estático, dinámico
   - régimen de evaluación: eager, lazy
   - sistema de tipo: safety, soundness
   - polimorfismo: ninguno, explícito, implícito, subtipos
   - recolección de basura: manual, automática

   Caracterice los lenguajes de programación que conoce, incluyendo al menos: C, Java, Scheme, (Common) Lisp, Haskell, y ML.