Lenguajes de Programación (2020/2)
Profesor: Federico Olmedo
Auxiliares: Bryan Ortiz Tomás Vallejos
Ayudantes: G. Iturra V. Illanes M. Rojas

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Examen

Para la resolución del examen recuerde que

- Debe entregar el desarrollo cada ejercicio por en un archivo separado
- Puede entregarlas en formato digital o un documento escaneado.

Ejercicio 1 **36 Pt**

Responda las siguientes consignas:

(a) [4 Pt] Haciendo una analogía con la taxonomía de funciones estudiada en el curso, responda la siguiente pregunta: ¿Qué significaría que en un lenguaje orientado a objetos, las clases fueran solo de primer orden?

(b) [4 Pt] A pesar de que C es un lenguaje (exclusivamente) *call-by-value*, éste permite (a través del mecanismo de punteros) que las modificaciones a parámetros formales de una función se propaguen hacia el contexto de la invocación de la función. ¿Cómo es esto posible? ¿Cómo podría lograrlo el lenguaje? Piense en el modelo del *store-passing style*.

(c) [3 Pt] Explique brevemente cuál es el problema de combinar evaluación perezosa con mutación.

(d) [3 Pt] Dé un programa en Racket que ilustre el problema anterior a través del uso de variables (recuerde el operador `set!`), suponiendo que Racket tenga una estrategia de evaluación perezosa.

(e) [4 Pt] ¿Por qué razonar sobre programas que modifican su estado es más difícil que sobre programas que no?

(f) [3 Pt] ¿Cuál es la diferencia entre TRO (*tail-recursion optimization*) y TCO (*tail-call optimization*)? Explique.

(g) [5 Pt] En Racket algunas formas como `(and e1 e2)` no son funciones. ¿Por qué no pueden serlo?

(h) [5 Pt] Considere la función `replace-or-add` definida en la clase 19 (diapositivas 19-20), que extiende un *store* con una asociación nueva, o actualiza una asociación ya existente (reemplazando la asociación antigua, en vez de enmascarándola). ¿Sería posible definir la misma función si se considerase una representación precedural de los *stores*? En caso negativo, explique por qué. En caso positivo, dé dicha definición (como una función de Racket).

(i) [5 Pt] En un lenguaje con substitución directa (es decir, no diferida), ¿son necesarias las clausuras si se desea preservar el *scope* estático?

Ejercicio 2 **9 Pt**

Considere la siguiente función de Racket que toma una lista de números y, explotando el hecho que $2^{a+S} = 2^a \cdot 2^S$, devuelve la potencia de 2 a la suma los elementos de la lista:

```
;; pow2-list :: listof (number) -> number
(define (pow2-list l)
    (match l
        ['() 1]
        [(cons x xs) (* (expt 2 x) (pow2-list xs)) ] ))
```

Recuerde que `(expt 2 x)` devuelve la potencia de `2` a la `x`.

(a) [3 Pt] Si se invoca a `pow2-list` con una lista de tamaño n, ¿la reducción va a requerir una pila de qué tamaño? (Con "tamaño" de la pila nos referimos al número de frames.) Justifique su respuesta.

(b) [6 Pt] ¿Sería posible construir una implementación en Racket de `my-pow2` que para su ejecución requiriese una pila de tamaño constante? En caso que no sea posible justifique porqué, y en caso de que sí sea posible, construya dicha implementación y explique en detalle por qué dicha implementación requeriría una pila de tamaño constante.

Ejercicio 3 **8 Pt**

Considere la macro en Racket:

```
(defmac (my-sum l r)
    (let ([v l])
        (if (zero? v) r (+ r v))))
```

y la expresión `e`:

```
(let ([v 5]) (my-sum 2 v))
```

(a) [4 Pt] Explique a qué expresión expande `e` en Racket y porqué.

(b) [4 Pt] ¿Qué otro resultado podría producir? ¿En qué circunstancias se obtendría este otro resultado?

Ejercicio 4 **7 Pt**

(a) [3 Pt] Ud recién descubrió un nuevo lenguaje que desconoce. ¿Qué puede decir del resultado del siguiente programa (la función `main` es el punto de entrada del programa)? Explique y detalle todo lo que considere relevante:

```
def y = 10;
def void foo(x)  { x = y; }
def void main() {
    def z = 5;
    foo(z);
    println(z);
}
```

(b) [4 Pt] Defina la función `(point x y)` tal que:

```
> (define p (point 2 3))
> (p 'getX)
2
> (p 'getY)
3
```

Hint: ¿qué tipo de valores retorna la función `point`? ¿qué mecanismo estudiado en clase permite que una función tenga "estado"?

Hint: `'getX` es un símbolo. Los símbolos pueden ser comparados con `eq?`.