Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3641 - Lenguajes de Programación I
Enero-Marzo 2016

Carnet:_____

Nombre:_____

Examen III

(40 puntos)

Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4	**Total**
18 puntos	6 puntos	8 puntos	8 puntos	**40 puntos**

Pregunta 1 - 18 puntos

Esta pregunta consta de seis (6) subpreguntas de selección, numeradas de 1.1 a 1.6. Cada una de las subpreguntas viene acompañada de cuatro posibles respuestas (a, b, c, d), entre las cuales sólo **una** es correcta. Ud. deberá marcar completamente y sin posibilidad de confusión la opción que considere correcta. Cada una de las subpreguntas tiene un valor de tres (3) puntos. Tres subpreguntas incorrectas eliminan una correcta.

1.1. Considere un lenguaje compilado con anidamiento de procedimientos. El lenguaje permite pasar arreglos por valor como parámetros, además de soportar *conformant arrays* de manera que una función pueda recibir arreglos de tamaños variables. Un procedimiento Q anidado n niveles dentro del procedimiento P es invocado pasándole un arreglo local a P. ¿Cuántas indirecciones son necesarias en el código de máquina generado para Q, hasta acceder al primer elemento del arreglo?

a) Exactamente n.
b) Exactamente n más un desplazamiento.
c) Al menos n.
d) Ninguna de las anteriores.

1.2. Considere la siguiente expresión válida en Haskell

```
head $ filter (==42) $ map grok ns
```

con `grok :: Num a => a -> a`. ¿Cuál de las siguientes afirmaciones es **cierta**?

a) Siempre consume espacio proporcional a la longitud de `ns`.
b) Siempre consume espacio constante.
c) Nunca demora tiempo proporcional a la longitud de `ns`.
d) Ninguna de las anteriores.

1.3. Siga considerando la expresión y condiciones de la pregunta anterior, pero ahora escrita en Racket

```
(car (filter (lambda (n) (eq? n 42)) (map grok ns)))
```

¿Cuál de las siguientes afirmaciones es **cierta**?

a) Siempre consume espacio proporcional a la longitud de `ns`.
b) Siempre consume espacio constante.
c) Nunca demora tiempo proporcional a la longitud de `ns`.
d) Ninguna de las anteriores.

1.4. En clase discutimos dos mecanismos para manejo de excepciones. ¿Cuáles de las siguientes afirmaciones son ciertas?

I. El que determina dinámicamente el manejador adecuado para cada instrucción, es más rápido en despacharla.

II. El que determina estáticamente el manejador adecuado para cada instrucción, es más lento en despacharla.

III. Los dos son igual de eficientes al despachar una excepción, pero uno es más rápido que el otro cuando no hay excepciones.

IV. Ambos requieren manejadores implícitos adicionales.

a) Sólo I, III y IV.

b) Sólo II, III y IV.

c) Sólo III.

d) Sólo IV.

1.5. En relación a las co-rutinas y los iteradores, se puede decir de manera general que:

a) Las co-rutinas y los iteradores, incluyendo recursión, sólo son factibles en lenguajes interpretados o de máquina virtual.

b) Son formas sintácticas derivadas del manejo generalizado de continuaciones.

c) Una co-rutina es un iterador capaz de ceder control selectivo entre varios llamadores.

d) Un iterador es una co-rutina que tiene múltiples puntos de entrada y de salida.

1.6. Considere las siguientes afirmaciones en relación a la programación orientada a objetos no jerárquica. ¿Cuáles son ciertas?

I. Un *mixin* es exactamente como una *interface*, pero que no se hereda.

II. Un *role* es un *mixin* con restricciones verificadas a tiempo de compilación.

III. Las técnicas de *duck-typing*, *interfaces*, *mixins* y *roles* suplen a la herencia múltiple.

IV. *Duck-typing* puede generar una excepción de «método inexistente» a tiempo de ejecución.

a) Sólo I, II y III.

b) Sólo II, III y IV.

c) Sólo II y III.

d) Sólo II y IV.

Pregunta 2 - 6 puntos

Considere el siguiente programa escrito en pseudocódigo

```
qux : int;

proc grok( foo, bar, baz : int )
  foo := foo + bar + baz + qux;
  print(qux);
  bar := foo + bar + baz + qux;
  print(qux);
  baz := foo + bar + baz + qux;
end

begin
  qux := 2;
  grok(qux,qux,qux)
  print(qux)
end
```

Ejecute el programa aplicando las reglas de pasaje de parámetros para cada uno de los casos especificados en la siguiente tabla. Suponga que el lenguaje pasa los parámetros en orden de izquierda a derecha.

Escriba en la columna **Salida del Programa** los resultados emitidos durante cada corrida.

Parámetro	**Pasaje por**	**Salida del Programa**
`foo`	Referencia	8
`bar`	Copia	8
`baz`	Referencia	26
`foo`	Copia	2
`bar`	Valor	2
`baz`	Referencia	8

Pregunta 3 - 8 puntos

Escriba las funciones

```
bfs :: Eq a => (a -> Bool) -> (a -> [a]) -> a -> Maybe a
dfs :: Eq a => (a -> Bool) -> (a -> [a]) -> a -> Maybe a
```

que implementan, respectivamente, la búsqueda BFS y DFS polimórficas sobre un espacio de soluciones, posiblemente infinito y con ciclos. Para ello, reciben tres argumentos:

- Un predicado `p` que indica si un estado particular es o no el estado que se está buscando.
- Un generador `g` que dado un estado particular, genera los posibles estados sucesores válidos sobre los cuales continuar la búsqueda.
- El estado inicial `s` a partir del cual iniciar la búsqueda.

Nota: para obtener los ocho (8) puntos *debe* escribir ambas funciones usando funciones de orden superior. Si las expresa con recursión de cola sólo recibira cuatro (4) puntos. Si las expresa con recursión directa sólo recibirá dos (2) puntos. Pero si Ud. es capaz de escribir ambas funciones como casos particulares de *una* sóla función auxiliar, escrita con funciones de orden superior y que a su vez es de orden superior, obtendrá cuatro (4) puntos extra.

Pregunta 4 - 8 puntos

Estoy pensando en un número. El número tiene diez cifras. Todas sus cifras son diferentes. Si observo las cifras de izquierda a derecha, noto algo interesante: si tomo las p primeras cifras de izquierda a derecha, resulta que el número formado por ellas es divisible entre p, y se cumple para $1 \leq p \leq 10$. Me pregunto si habrá más de un número que cumpla esto.

- **(6 puntos)** Escriba el predicado `n/1` que triunfe si su único argumento `N` unifica con un número entero que cumple con las restricciones indicadas.
- **(2 puntos)** Escriba el predicado `acertijo/0` que siempre triunfe, y aproveche *backtracking* explícito para encontrar todas las soluciones posibles, imprimiendo una por línea.

Nota: Sólo puede utilizar predicados que estén en Prolog estándar. Si Ud. «conoce» algún predicado que parece útil, pero no sabe si está en Prolog estándar, impláantelo.

La siguiente solución representa un balance aceptable en cuanto a la «astucia» con la cual generar la solución y la generalización de la verificación.

```
assign_digits([],_).
assign_digits([N|Ns],Ds) :-
        select(N,Ds,Dss),
        assign_digits(Ns,Dss).

build(Digits,N)       :- build(Digits,0,N).
build([D|Ds],Acc,Res) :- NewAcc is Acc*10 + D,
                         build(Ds,NewAcc,Res).
build([],Acc,Acc).

check(Ds,11).
check(Ds,Divisor) :-
        P is 10 - Divisor,
        length(Suffix,P),
        append(Prefix,Suffix,Ds),
        build(Prefix,Check),
        0 is Check rem Divisor,
        Divisor1 is Divisor + 1,
        check(Ds,Divisor1).

n(N) :-
        Number = [_,_,_,_,_,_,_,_,_,_],
        Digits = [9,8,7,6,5,4,3,2,1,0],
        assign_digits(Number,Digits),
        check(Number,1),
        build(Number,N).

acertijo :- n(N), write(N), nl, fail.
acertijo.
```