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

Carnet:_____

Nombre:_____

Examen I

(25 puntos)

Pregunta 1	Pregunta 2	Pregunta 3	**Total**
12 puntos	6 puntos	7 puntos	**25 puntos**

Espacio para contestar la pregunta (3)

Pregunta 1 - 12 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 dos (2) puntos. Tres subpreguntas incorrectas eliminan una correcta.

1.1. El diseño de un lenguaje de programación establece que en todas sus implantaciones los números enteros deben tener precisión arbitraria exacta ilimitada. Tal decisión de diseño puede favorecer o no las siguientes características del lenguaje:

I. La eficiencia de ejecución de programas escritos en el lenguaje.

II. La portabilidad de la implantación del lenguaje.

III. La portabilidad de los programas escritos en el lenguaje.

¿Cuáles **perjudica**?

a) I y II.

b) I y III.

c) II y III.

d) Todas.

1.2. Considere un lenguaje compilado, con alcance dinámico, y sin funciones anidadas. Entonces es cierto que:

a) Su compilador maneja una tabla de símbolos del estilo Tabla Central de Referencias.

b) Para todo programa en ejecución, la cadena estática es idéntica a la cadena dinámica.

c) Es imposible escribir un *debugger*.

d) Ninguna de las anteriores.

1.3. Considere una variable estática local a un procedimiento. En ese caso:

a) Se elabora en la pila de ejecución.

b) Se elabora al iniciar el programa.

c) Se accede a través de la cadena estática.

d) Ninguna de las anteriores.

1.4. Considere un lenguaje imperativo, de propósito general, con alcance estático. ¿Cuál de las siguientes verificaciones **no** puede completarse enteramente de forma estática?

a) Verificar que un bloque de instrucciones nunca será ejecutado.

b) Verificar que toda variable recibe valores cónsonos con su tipo.

c) Verificar que toda variable siempre es inicializada.

d) Verificar que una variable nunca es usada.

1.5. En relación a módulos como administrador y módulos como tipo podemos afirmar:

a) Los primeros son similares a las librerías tradicionales,
y los segundos son similares a la orientación a objetos.

b) Los primeros necesitan tener espacio de nombre cerrado, pero los segundos pueden operar con espacio abierto o cerrado indistintamente.

c) Se diferencian operacionalmente, pues el código ejecutable final de los módulos como administrador es más complejo.

d) Son indispensables para que un lenguaje construya programas por compilación separada.

1.6. El ambiente de ejecución de un lenguaje con almacenamiento en *heap* requiere un manejador de memoria para administrar el espacio adicional. Considere las siguientes afirmaciones en relación a cualquier manejador de memoria:

I. *Best Fit* siempre es mejor que *First Fit* para reducir la fragmentación externa.

II. Variar el tamaño de los bloques reduce la fragmentación interna.

III. La compactación de memoria contribuye a reducir la fragmentación externa.

¿Cuáles afirmaciones son *ciertas*?

a) I y II solamente.

b) II y III solamente.

c) I y III solamente.

d) Todas son ciertas.

Pregunta 2 - 6 puntos

Considere el siguiente programa escrito en pseudocódigo:

```
int foo = 2
int bar = 3

proc B(proc wee, int bar)
  if bar > 3
    B(wee,bar-2)
  wee(bar)
  print(bar)

proc A(int baz, int foo)
  proc fudge(int qux)
    bar := foo + bar + qux

  int bar = baz
  B(fudge,foo)
  print(bar)

main
  A(foo,4)
  print(bar)
end
```

Ejecute el programa aplicando las reglas de alcance y construcción de clausuras para cada uno de los casos especificados en la siguiente tabla. Escriba en la columna **Salida del Programa** los resultados emitidos durante cada corrida.

Alcance	**Asociación**	**Salida del Programa**
Estático	Profunda (*Deep*)	
Dinámico	Profunda (*Deep*)	
Dinámico	Superficial (*Shallow*)	

Pregunta 3 - 7 puntos

Ud. está usando Haskell para implantar el prototipo de un manejador de memoria. En este prototipo, se emplean los tipos de datos

```
type Block    = (Int,String)  -- (Capacidad,Bytes)
type FreeList = [Block]       -- Bloques en cualquier orden
```

para representar los bloques de almacenamiento en el *heap* indicando su tamaño y la secuencia de bytes almacenada allí, así como la lista de bloques libres disponibles. Se considera que un bloque está vacío si está completamente lleno de asteriscos, i.e.

```
(42,'******************************************')
```

es un bloque vacío de tamaño 42. Entonces, implante la función

```
malloc :: FreeList -> Int -> (Block,FreeList)
```

que permite reservar un bloque de memoria. El primer argumento corresponde a la lista de bloques libres, de la cual se pretende tomar un bloque cuyo tamaño permita satisfacer lo requerido por el segundo argumento, siguiendo la estrategia *First Fit*. Así, la función retornará un bloque vacío del tamaño requerido, y la lista de bloques libres después de la reserva.

Si se invoca la función `malloc` sobre un `FreeList` en el cual no hay ningún bloque que permita satisfacer el requerimiento, es necesario ampliar la lista de bloques libres con cuatro (4) bloques todos del mismo tamaño, que debe ser exacto para satisfacer el requerimiento – esto es, debe simularse la invocación de `sbrk` al sistema de operación cuando sea necesario.

Nota: puede usar cualquier función del *Preludio* y escribir funciones auxiliares si le conviene.

```
empty :: Int -> Space
empty s = replicate s '*'

malloc :: EmptyList -> Int -> (Block,EmptyList)
malloc [] s     = malloc (sbrk 4 s) s
malloc (b:bs) s = if s <= fst b then (b,bs)
                                else (b', b : bs')
                  where (b',bs') = malloc bs s

sbrk 0 _ = []
sbrk n s = (s,empty s) : sbrk (n-1) s
```