Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-3641 - Lenguajes de Programación I
Enero-Marzo 2013

Carnet:_____

Nombre:_____

Examen I

(25 puntos)

Pregunta 1	Pregunta 2	**Total**
18 puntos	7 puntos	**25 puntos**

Pregunta 1 - 18 puntos

Esta pregunta consta de nueve (9) subpreguntas de selección, numeradas de 1.1 a 1.9. 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 o, si desea no contestar la subpregunta, marcar completamente y sin posibilidad de confusión la última opción (e) que indica que Ud. prefiere omitir la respuesta.
Cada una de las subpreguntas tiene un valor de dos (2) puntos. Tres subpreguntas incorrectas eliminan una correcta. Las subpreguntas omitidas (marcadas en la opción e) no suman ni restan puntos.

1.1. La definición de algunos lenguajes, como C, C++ y Pascal, señala que la liberación de memoria en *heap* debe ser realizada explícitamente por el programador. Esta decisión de diseño favorece:

a) La portabilidad de los programas escritos en el lenguaje.

b) La eficiencia de construcción y mantenimiento de los programas escritos en el lenguaje.

c) La eficiencia de ejecución de los programas escritos en el lenguaje.

d) Ninguna de las anteriores.

e) No sabe / No contesta.

1.2. En los lenguajes con alcance estático, el registro de activación para cada subrutina invocada almacena apuntadores que permitan acceder a las cadenas estática y dinámica. En relación a estas cadenas *siempre es cierto* que:

a) Puede haber elementos en la cadena dinámica de un registro que no estén presentes en la cadena estática del mismo registro.

b) La cadena estática y dinámica para un registro particular divergen a partir del registro de activación del llamador.

c) Son manejadas en memoria dinámica, pues están constituidas por apuntadores generados a tiempo de ejecución.

d) Ninguna de las anteriores.

e) No sabe / No contesta.

1.3. Considere un lenguaje que solamente tiene alcance dinámico y no tiene variables globales ni datos almacenados estáticamente. Esto es, el diseñador e implantador decidieron que el lenguaje solamente utilizará almacenamiento en pila de ejecución y en *heap*. Entonces

a) El lenguaje no puede soportar constantes explícitas ni implícitas, *debugger*, *profiler* y ninguna característica que necesite almacenar información estáticamente.

b) La pila de ejecución hace las veces de Lista de Asociación.

c) El lenguaje sólo puede implantarse como interpretador.

d) La pila de ejecución hace las veces de Tabla Central de Referencia.

e) No sabe / No contesta.

1.4. Considere una variable almacenada en espacio estático. ¿Cuál de las siguientes afirmaciones *podría* ser cierta?

a) Su tiempo de vida corresponde con el tiempo de ejecución del programa, pero sólo es accesible dentro de un alcance local.

b) Su tiempo de vida comienza cuando se usa efectivamente por primera vez en el flujo de ejecución, y es alcanzable en todo el programa.

c) Su tiempo de vida tiene interrupciones ocasionadas por la aparición de alcances conteniendo otras variables con el mismo nombre.

d) Ninguna de las anteriores.

e) No sabe / No contesta.

1.5. Los módulos como tipo:

a) No pueden ofrecer abstracción completa cuando operan con alcance abierto.

b) Si se usan más de una vez en el mismo programa, el compilador se ve en la necesidad de repetir el código de máquina generado.

c) Están obligados a exportar la representación interna del tipo de datos.

d) Permiten considerar al tipo de datos como poseedor de comportamiento, en lugar de ser simples valores manipulables.

e) No sabe / No contesta.

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. La fragmentación interna puede reducirse, pero es imposible eliminarla de forma general.

II. El método *Best Fit* aumenta la fragmentación interna, pero es muy eficiente a la hora de encontrar bloques libres.

III. La fragmentación externa puede combatirse compactando bloques de memoria adyacentes durante la liberación de memoria.

IV. Si se usan múltiples listas de bloques libres, el método Fibonacci es exactamente lo mismo que el método *Buddy* en lo que respecta a disminuir la fragmentación interna.

¿Cuáles afirmaciones son ciertas?

a) I y III solamente.

b) II y III solamente.

c) I, III y IV solamente.

d) III y IV solamente.

e) No sabe / No contesta.

1.7. Considere el siguiente programa escrito en pseudocódigo:

```
var foo : int;

proc woot(int qux)
  int bar;

  proc what ( baz : int )
    foo := foo + bar + baz
  end;

  proc wait ( p : proc, tf : int )
    int foo;
    foo := tf;
    if tf < 42
      wait( what, 3 * foo )
    else
      p( tf )
    endif
    print foo
  end;

  bar := qux * 2
  wait( what, bar )
end;

main
  foo := 7;
  woot( foo )
  print( foo );
end;
```

¿Qué imprime el programa si opera con alcance estático y asociación profunda (*deep binding*)?

a) 42 14 7

b) 63 14 7

c) 42 14 63

d) 7 63 42

e) No sabe / No contesta.

1.8. Continúe con el programa dado en la subpregunta (1.7), ¿qué imprime si opera con alcance dinámico y asociación profunda (*deep binding*)?

a) 42 70 7

b) 98 14 42

c) 42 14 7

d) 14 42 70

e) No sabe / No contesta.

1.9. Continúe con el programa dado en la subpregunta (1.7), ¿qué imprime si opera con alcance dinámico y asociación superficial (*shallow binding*)?

a) 42 70 7

b) 98 14 7

c) 98 14 42

d) 42 14 63

e) No sabe / No contesta.

Pregunta 2 - 7 puntos

En el lenguaje Java, las variables de tipo arreglo almacenan en realidad referencias a arreglos. Esto hace que las variables de tipo arreglo y los arreglos sean objetos independientes entre si, entre los cuales se pueden crear asociaciones (*bindings*). Tal independencia se ve reflejada en los tiempos de vida de éstos, de manera tal que el tiempo de vida t_v de una variable de tipo arreglo y el tiempo de vida t_a de un arreglo con el que la variable pueda ser asociada (*bound*) no tienen por que ser el mismo. Esto es, puede ser que t_a incluya a t_v o puede ser que t_v incluya a t_a.
Se desea que Ud. escriba dos pequeños programas en Java que muestren ejemplos de solapamiento. En cada programa debe haber una asociación (*binding*) entre una variable `v` de tipo arreglo y un arreglo, de manera tal que:

- En el primer programa, la variable `v` se elabora *estrictamente* antes que el arreglo, y se destruye *estrictamente* después que el arreglo.
- En el segundo programa, el arreglo se elabora *estrictamente* antes que la variable `v`, y se destruye *estrictamente* después que la variable.

Aparte de la variable `v` y el arreglo en cuestión, sus programas pueden tener objetos adicionales (variables, métodos, etc.) que considere necesarios para lograr el comportamiento solicitado. Tenga en cuenta que en Java las variables locales a los métodos se elaboran al inicio de la ejecución de ese método. Justifique brevemente el funcionamiento de cada programa.