Trabajo Práctico N° 04

Administración de Memoria

Recomendaciones Generales e Información

Abreviaciones empleadas durante el práctico:

RA: Registro de Activación
CR: Contador de Referencia
GC: Garbage Collector

Al resolver los gráficos es fundamental que no presenten un único gráfico final sin los pasos intermedios ya que no hay forma de deducir la secuencia y la forma en que se fue armando.
Presentar un gráfico distinto para cada paso.

Algoritmo de Recorrido y Marcado local:

A partir del puntero borrado
1. Recorrer los bloques decrementando los contadores y marcando los nodos como posible Garbage
2. Se recorre nuevamente buscando nodos con cuenta diferente de cero y se desmarcan y actualizan los contadores de sus hijos
3. Todos los bloques marcados se mueven a lista de libres.

Los métodos de Garbage Collection analizados en la clase teórica son suficientes para resolver el práctico, sin embargo, se provee Información adicional para quien quiera profundizar conceptualmente en el tema y para verificar las implementaciones de Garbage Collection en diferentes lenguajes

Libro general sobre Garbage Collection

- Jones, R. E. (1996). *Garbage collection: algorithms for automatic dynamic memory management*. John Wiley and Sons. (Contador de referencias, Marcado y borrado, entre otros)
- Jones, R., Hosking, A., & Moss, E. (2016). *The garbage collection handbook: the art of automatic memory management*. CRC Press.

Marcado y borrado local (el algoritmo presentado en clase es una simplificación de estos):

- Martínez, A. D., Wachenchauzer, R., & Lins, R. D. (1990). Cyclic reference counting with local mark-scan. *Information Processing Letters*, *34*(1), 31-35.
- Lins, R. D. (1992). Cyclic reference counting with lazy mark-scan. *Information Processing Letters*, *44*(4), 215-220.

Documentación Garbage Collection en algunos lenguajes:

- Java
- C#
- Python 3
- Go

Ejercicio 1

Considere el siguiente fragmento del programa:

```
int *a, *b, *c, *d, *e;
int x =10 ;
a = (int *) malloc (0x50);
b = (int *) malloc (0x100);
c = (int *) malloc (0x200);
d = (int *) malloc (0x300);
c = &x ;
b = d ;
gc
free(b);
e = (int *) malloc (0x100);
a = c;
d = e;
gc;
a = (int *) malloc (0x100);
b = a;
c = b;
free(d);
```

- La sentencia ***gc*** representa la ejecución del algoritmo de garbage collector para lenguajes tipo **Algol**.
- Considere además que el RA que contiene a las variables comienza en la dirección $\mathbf{2000}_h$ y que los offsets son a($\mathbf{10}_h$), b($\mathbf{14}_h$), c($\mathbf{18}_h$) y así sucesivamente de a 4 bytes para cada variable declarada en el RA.
- El heap se encuentra desde la dirección $\mathbf{B800}_h$ (0xB800) hacia direcciones menores.
- Suponga que se usa el algoritmo de first fit para asignar bloques de memoria.

a) Clasificar todas las variables, de acuerdo a su tipo de almacenamiento. Justificar sus respuestas.
b) Dibujar un esquema que represente todos los elementos anteriores inicialmente (hasta malloc $\mathbf{300}_h$) en la memoria indicando las direcciones de cada elemento.
Incluya los punteros de la pila al heap y el contenido de las variables en el RA.
c) Dibujar e indicar, paso a paso, cómo se modifican las variables
d) Indicar, luego de todas las instrucciones, qué bloques (las direcciones) quedan usados y no son garbage en el heap.
e) Indicar la lista de los bloques libres (las direcciones) en el heap.
f) Indicar qué bloques quedan como garbage y cuáles como fragmentación.
g) ¿Cuál es el contenido de cada una de las variables definidas en la pila luego de la última sentencia del programa?

Ejercicio 2

Dado el siguiente fragmento de programa en un lenguaje tipo **Algol** que utiliza el algoritmo de contadores de referencia. Considere que el RA que contiene las variables comienza en la dirección $\mathbf{2A00}_h$, que el offset para **pa** es ($\mathbf{20}_h$) y así sucesivamente de a 4 bytes para cada variable integer. El heap decrece desde la dirección $\mathbf{B800}_h$.

```
int *pa,*pb;
int a,b,c;
Vector<int> d;
int array e[2:6] = 5;
b = e[3];
a = b;
d= (10,20,30);
pa=malloc(0x100);
pb=malloc(0x200);
pa=pb;
free(pb);
pa=malloc(0x300);
c=b;
d= (10,20,30,40,50,60,70);
pb=pa;
```

a) Clasificar todas las variables presentes en el código de acuerdo a su almacenamiento.
b) Dibujar tres esquemas de memoria: uno con todos los elementos inicialmente en la memoria que incluya los punteros de la pila al heap, otro con las modificaciones paso a paso de las asignaciones y otro que muestre cómo queda al final la memoria.
c) Indique qué instrucciones, a lo largo del programa, produjeron garbage, cuáles fragmentación y cuáles punteros colgados. Explique sobre qué bloques o punteros se produjeron.

Ejercicio 3 (Plan 2011)

Considere el siguiente fragmento del programa que utiliza el algoritmo de contadores por referencia:

```
int *a, *b, *c, *d;
a = (int *) malloc (0x32);
b = (int *) malloc (0x64);
c = (int *) malloc (0x96);
d = (int *) malloc (0xC8);
c = d ;
c = (int *) malloc (0x12C);
d = b;
free(a);
a = c;
free(b);
```

a) Clasificar cada una de las variables declaradas de acuerdo a su tipo de almacenamiento e indicar cuáles podrían producir garbage, cuáles fragmentación y cuáles punteros colgados. Justificar sus respuestas.
b) Indicar, paso a paso para cada instrucción, cómo se modifican los contadores de referencia en el heap y las variables indicando cuáles producen garbage, cuáles fragmentación y cuáles punteros colgados.

Ejercicio 4

Considere las siguientes sentencias:

```
int point;
int *ppoint;
ppoint = malloc(100);
```

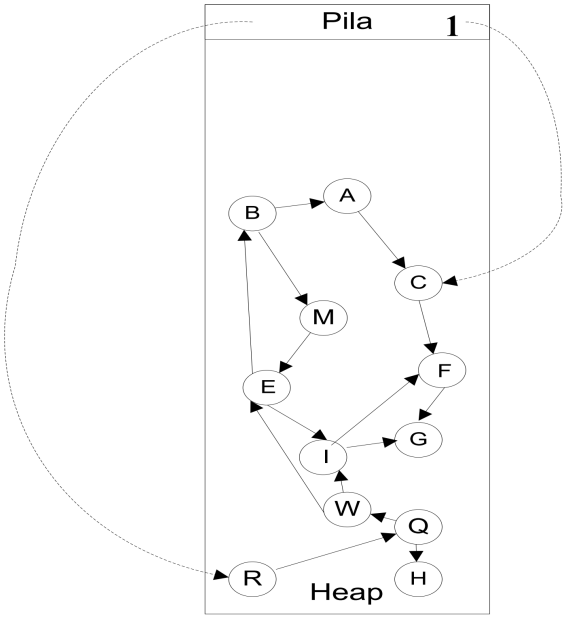
Donde **point** y **ppoint** son dos variables almacenadas en la pila de RA.
Si el lenguaje usa el algoritmo de garbage collection basado en la inserción de CR a los datos apuntados:
¿Es cierta la siguiente afirmación?

No es necesario agregar un CR ni en **point** ni en **ppoint**

Justifique la respuesta.

Ejercicio 5

Se tiene un lenguaje que realiza el garbage collection usando contadores de referencia. Considere el siguiente ejemplo:



Suponga que se elimina la referencia del bloque usado **R** al bloque usado **Q**.

a) ¿Qué elementos del heap serán declarados garbage y pasados a la lista de bloques libres? ¿Por qué?
b) ¿Qué elementos del heap serán garbage pero esta situación resultará invisible en la técnica de CR? ¿Por qué?

c) ¿Qué elementos del heap podrán ser declarados garbage usando la técnica de marcado y borrado de marcas local?
Detalle todos los estados intermedios
d) Realice un nuevo gráfico que muestre cómo quedaría el ejemplo luego de aplicar la técnica de marcado y borrado de marcas local.
e) ¿Cuál es la función que cumple para este algoritmo la referencia a **C** desde la pila?

Ejercicio 6

Dado el siguiente código en lenguaje **Java**:

```
class B {
   public B b2;
}
class A {
   public B b1;
}
public class Main {
   static public int main() {
1          A a1 = new A();
2          A a2 = new A();
3          B b2 = new B();
4          a1.b1 = new B();
5          a2.b1 = new B();
6          a2.b1 = a1.b1;
7          a1.b1.b2 = new B();
8          a1.b1.b2.b2 = a1.b1;
9          a1=a2;
10         a2.b1.b2 = new B()
11         a2.b1.b2.b2 = a2.b1;
12         b2=a2.b1;
13         a2 = new A();
14         a1 = new A();
   }
}
```

La función `new Nodo` cumple la función de la llamada `(Nodo*) malloc(sizeof(Nodo));`

a) Realizar un diagrama de memoria marcando los efectos que tiene cada instrucción sobre el diagrama.
b) Describir cada una de las ejecuciones del garbage collector aplicando el método de marcado y borrado local, indicando en qué instrucción se ejecuta, cuáles nodos son marcados como garbage, los CR de cada nodo e indicar qué nodos quedan en memoria al finalizar.
c) Indicar en cuáles instrucciones en las que se cambia un puntero a una estructura del heap, una parte de dicha estructura no se marca como garbage.
d) Si el lenguaje no tuviese garbage collection, con este código ¿se podrían producir resultados de las estructuras que son garbage?

e) Si el lenguaje tuviese la característica de hacer que un bloque del heap apunte a una dirección de la pila ¿ésto podría producir, potencialmente, más garbage?

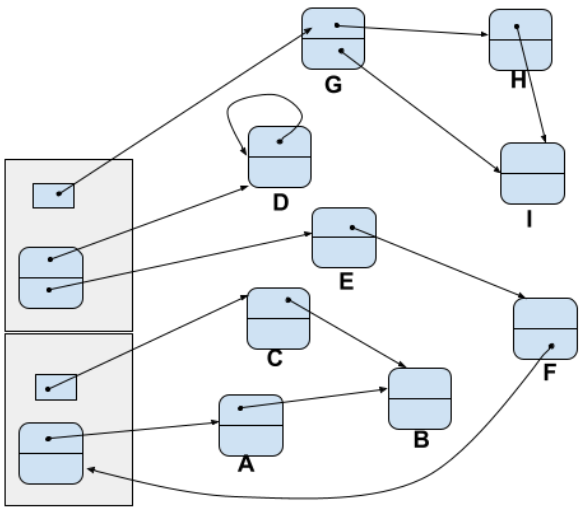
Ejercicio 7

Dado el siguiente código en un lenguaje que distingue entre los objetos y punteros a ellos:

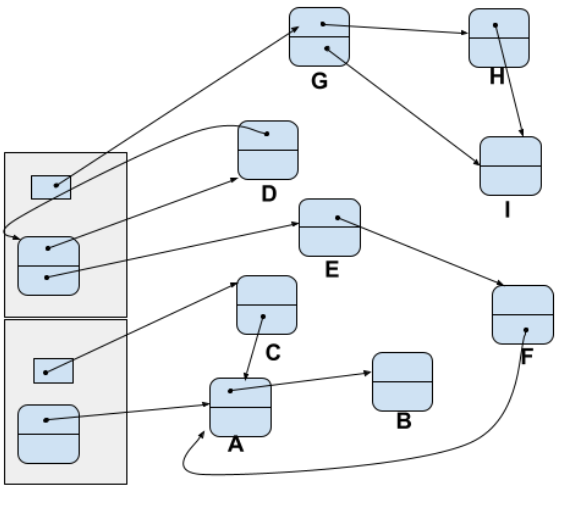
```
1   class A {
2   public:
3       A *dos
4       A *uno
5   }
6   A a1
7   A *a2
8   void principal()
9    {
10      A a3
11      A *a4
12      a1.uno = new A()
13      a1.uno.uno = new A()
14      a2 = new A()
15      (*a2).uno = a1.uno.uno
16      a3.uno = new A()
17      a3.uno.uno = &(a3.uno)
18      a3.dos = new A()
19      a3.dos.uno = new A()
20      a3.dos.uno.dos = &a1
21      a4 = new A()
22      (*a4).uno = new A()
23      (*a4).dos = new A()
24      (*a4).uno.uno = (*a4).dos  //hasta aquí
25      a1.uno = null
26      a2 = null
27      a3.uno = null
28      a3.dos = null
29      a4 = null
30  }
```

a) Determinar cuál diagrama se corresponde al código en el instante que finaliza la ejecución de la instrucción comentada con **"hasta aquí"**. ***Aclaración:*** *En los diagramas de objetos de la clase A, el campo "uno" está arriba y el campo "dos" está abajo.*

D1



D2



b) Al diagrama correcto colocarle, para cada puntero, el número de instrucción asociado a su creación.

c) Considerando que el lenguaje posee el **método de marcado y borrado local**, completar la siguiente tabla con la ejecución de todo el programa, indicando *para cada instrucción* ***(de la 12 en adelante)***:

- *El nodo del heap al cual crea o afecta su contador de referencias* ***(en caso de no afectar ninguno colocar "-")***
- *Cuánto vale el contador antes y después de la instrucción*
- Si, como consecuencia de la instrucción, el nodo se convierte en garbage (indicar SI/NO)
- *Si una instrucción provoca efectos en más de un nodo, repetir la instrucción por cada nodo que afecte secuencialmente.*
  Ejemplo: si la instrucción 37 provoca que se decremente el contador del nodo Z de 3 a 2, se coloca: 37, Z, 3, 2, SI y luego si en el segundo paso se llega a ese nodo, se coloca 37, Z, 2, 2, NO.

Insrucción	Nodo	Contador de referencias antes	Contador de referencias después	Garbage

d) Si se reemplaza el algoritmo de garbage collector de marcado y borrado local por el de marcado y borrado, invocándolo al final del programa ¿cuáles nodos serán marcados como garbage al finalizar su ejecución? **Justifique su respuesta.**

Ejercicio 8

Se tiene el siguiente programa en lenguaje **JAVA**. Se pretende analizar el comportamiento de las variables inmutables del programa.

```
public class Inmutables{
   public static void main(String[] args){
        integer a = 1;
        integer b = 2;
        integer c = a; [1]
        system.out.println(a);
        system.out.println(b);
        system.out.println(c);
        a = 3;
        system.out.println(a);
        system.out.println(b);
        system.out.println(c);
        [2]
   }
}
```

a) ¿Qué imprime este programa y por qué? Realice, para justificar, un gráfico de las variables en memoria con sus respectivos valores.

b) ¿Cómo cambiaría la instrucción **[1]** para que produzca garbage, sin borrar ni agregar variables ni líneas?

c) Si a los cambios del inciso anterior se agregan las siguientes sentencias al programa en **[2]** ¿cuántos bloques del heap quedarían en total como garbage y por qué?

```
for(int i=0;i<5;i++) {
    a=a+1;
    System.out.println(a);
}
```

Ejercicio 9

Se tiene un lenguaje con variables inmutables (variables dinámicas con nombre cuyas celdas no pueden ser modificadas y poseen almacenamiento en el heap durante toda la ejecución del programa). Responder si las siguientes afirmaciones son verdaderas o falsas, fundamentando las respuestas:

a) Considerando que el tiempo de vida de las celdas de estas variables es de todo el programa, esto contribuye a disminuir la existencia de garbage.
b) Un puntero a estas variables puede convertirse en un puntero colgado
c) Una asignación a una variable inmutable, dentro de una iteración, provoca la creación sucesiva de garbage
d) Las variables inmutables pueden ser sólo de tipo string

Ejercicio 10 (Plan 2011)

Responder si las siguientes afirmaciones son **verdaderas** o **falsas**, justificando las respuestas:

a) Una variable dinámica anónima que apunta a una variable semiestática que es eliminada, provoca que el contador de referencia de la variable semiestática se decremente en uno.
b) Los algoritmos de marcado y borrado y el de marcado y borrado local se ejecutan cada vez que se modifica el valor de un puntero.
c) Las variables dinámicas con nombre pueden producir garbage y pueden ocupar bloques no contiguos en el heap
d) Es posible que se produzcan punteros colgados tanto desde como hacia la pila de registros de activación
e) Si las variables semi dinámicas estuviesen en el heap aumentaría la producción de garbage
f) Si las variables semi dinámicas estuviesen en el heap, la ejecución de las unidades sería más lenta
g) Las asignaciones por referencia pueden favorecer a crear punteros colgados

Ejercicio 11

Dado el siguiente código en el lenguaje Rust:

fn main() {
let x = 1;
x = 2;
let mut y = 2;
y = 3;
let p1 = &x;
let p2 = &y;
	a) Compilarlo en un compilador local de Rust o uno online
	b) Identificar cada uno de los errores que se producen si son por: intento de modificación de una variable mutable, intento de asignar una dirección de una variable inmutable a un puntero a variable mutable, intento de préstamo (borrowing) de una variable como mutable, ya prestada como inmutable,

let mut p3 = &x;
let mut p4 = &y;
let p5 = &mut x;
let mut p6 = &mut x;
let p7 = &mut y;
let mut p8 = &mut y;
*p7 = 5;
*p8 = 6;
*p3 = 3;
*p4 = 4;
}
	Intento de asignar a una variable inmutable a través de una referencia.
	c) Comentar las líneas que sean necesarias para que el programa compile.