# CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN

# TRABAJO INTEGRADOR 2017

# Segunda Parte


**INTEGRANTES:**

- **CARACOTCHE ROMINA ANTONELLA, 11105/1**
- **SALGADO IVAN, 11823/6**
- **STANCICH DAVID ALFREDO, 11429/9**


**GRUPO:** 21
**LENGUAJES:** Matlab - Java - C

**AYUDANTE:** José
**HORARIO:** Lunes 18hs

**BIBLIOGRAFÍA**


- https://lsi.ugr.es/curena/doce/lp/tr-11-12/lp-c03-impr.pdf
- https://icsdonald.files.wordpress.com/2012/02/ligadura-de-una-variable.pdf
- https://www.javatpoint.com/
- http://www.aprenderaprogramar.com/
- https://www.tutorialspoint.com/matlab/


# Mathlan

## Correcciones de la entrega #1:

En Java por defecto las variables normales se inicializan con basura, en caso de que se cree una variable y se trate de usar sin antes haberla inicializado producirá un error en tiempo de compilación.
Los arreglos (que en Java son objetos) cuando se los instancia, los elementos de sus posiciones si se inicializan con valores por defecto. Se los inicializa con los siguientes valores:

- byte - short - int - long => 0
- float - double => 0.0
- boolean => false
- char => /n
- Objeto => null

En C las únicas variables que se inicializan por defecto son las globales y las estáticas.

## 2da Parte (Fecha Final de entrega: 19/5):

**D. Determine y explique los atributos de las variables de su lenguaje así como su ligadura y estabilidad. Ejemplifique comparando con los lenguajes asignados.**

**E. Determine los tipos de variables de acuerdo al momento de ligadura con el l-valor de su lenguaje. Ejemplifique la forma en que simularía (en caso de no poseerla) una variable de tipo estática. En el caso en que cuente con este tipo de variables ejemplifique la diferencia con una variable global.**

**F. Realice una comparación respecto de la utilización de parámetros y los diferentes modos que implementa. Ejemplifique. Utilice el compilador de los lenguaje asignados que se encuentra en la siguiente página http://www.tutorialspoint.com/codingground.htm para realizar los ejemplos planteados.**

**D)** En Mathlan las variables están compuestas por los siguientes atributos:

- **Identidad**
- **Nombre**
- **Localización de memoria**
- **Valor**
- **Tipo**
- **Tiempo de vida**
- **Alcance**

Identidad de las variables:

- Cada variable del programa tiene asociada una identidad única.
- No puede cambiar durante el tiempo de vida de la variable.
- Existe una función que asocia a cada identidad una localización de memoria.
- La identidad puede implementarse mediante el tipo puntero, mediante un índice en una tabla de variables (como en Java) o mediante el nombre de la variable.

Nombre de las variables:

- Es un identificador (una secuencia de dígitos y/o letras).
- Representa a la variable en el programa fuente.
- Permite al programador aportar información sobre el significado del valor almacenado.
- Pueden existir variables con varios nombres (a esos nombres se le llaman alias).

Localización de memoria:

- Se puede calcular a partir de la identidad.
- Identifica donde está alojado el valor.
- Puede cambiar durante el tiempo de vida de la variable.
- Debe permitir al programador abstraerse de este valor.

Tipo de las variables:

- Determina el conjunto de valores que puede tomar la variable, y las operaciones que pueden hacerse sobre ella.
- Mathlan fuerza a asociar un tipo único a cada variable.

Tiempo de vida:

La ejecución de un programa ocupa un intervalo de tiempo:

- El tiempo de vida de una variable es el intervalo o intervalos de tiempo durante los cuales la variable existe.
- Los intervalos están normalmente incluidos en el intervalo de tiempo de ejecución del programa.
- Una variable existe durante la ejecución cuando se puede acceder a su valor, en el sentido de que se puede garantizar que dicho valor está almacenado en su localización usando la representación asociada a su tipo.

Alcance:

- Es la porción o porciones del programa en los cuales la aparición del nombre de la variable es correcta y hace referencia a dicha variable, a sus atributos o a su valor.
- Define en qué secciones de código una variable estará disponible. Fuera de este ámbito, una variable no podrá ser accedida (no existe).

En Java y en C el momento de ligadura es en compilación. Si tenemos una variable entera y le indicamos que es un Integer en compilación, ya sabemos que esa variable va a ser siempre entera. En ejecución se infiere el tipo de acuerdo a la expresión, y en este caso se ligaría con el tipo en tiempo de ejecución. Se utiliza en todos los métodos de instancia de Java que no son privados ni final.
Es por esto que nuestro lenguaje también va a tener ligadura estática según el tipo declarado del objeto al que se manda el mensaje. Java utiliza los métodos de clase y los métodos de instancia que son privados o final (ya que estos últimos no pueden ser sobrescritos).

.
**E)** Tipos de Variables según su momento de L-valor:

- ➔ **Estática:** Son las variables que se definen con la palabra clave “static” y su tiempo de vida se extiende durante la ejecución de todo el programa. Este tipo de variables las implementan C y Java. En C si le ponemos el calificador static sabemos que esa variable está en memoria y se asoció con su dirección de memoria en tiempo de compilación.

- ➔ **Automática:** Son las variables que no tienen ningún clasificador que las altere.

- ➔ **Dinámica:** Son las variables apuntadas en la memoria heap. Este tipo de variable las implementa java.

La principal diferencia es que toda variable global por ser automática va al segmento de datos, en cambio una variable estática va al segmento de código. Al ir al segmento de datos cada vez que se desaloca algo muere; en cambio, al estar en el segmento de código mientras esté el código en memoria la variable estática va a seguir alocada en memoria por lo tanto va a tener mayor tiempo de vida que una variable global.
Otra diferencia es que una variable estática es sensible a la historia, es por esto que puede pasar que una variable global termine teniendo menos alcance que una variable estática.
En C una variable estática significa que se va a alocar en memoria en tiempo de compilación, mientras que en java simula una variable global (variables de clase). Si dentro de una clase ponemos public static eso hace que si una instancia de esa clase modifica a la variable lo hace para todas las instancias de esa clase (variable de clase).

**Archivo #1**

```
int a;
static int b;

void func() {
    static int aux;
    char car;
}

int main() {
    return 0;
}
```

**Archivo #2**

```
int k;

void func() {
        float h;
}
```

El alcance de la variable “a” se extiende en todo el archivo #1, al igual que la variable “k”, pero en el archivo #2, al menos que con una declaración especial esta pueda extender su alcance hacia otros archivos.
El alcance de la variable “b” y “aux” es en el archivo que se encuentra definida y no puede extender su alcance hacia otro archivo; pero su tiempo de vida es durante todo el programa.
La variable “car” es una variable interna, una vez terminada la función esta se desaloja de memoria.

## F) Java

En este lenguaje los parámetros de los tipos primitivos (byte, short, int, long, float, double, char y boolean) siempre se pasan por valor.
Java no cuenta con pasaje por referencia, y tampoco pasa objetos como parámetros, sino copias de las referencias a esos objetos, esto quiere decir que cuando se manda la copia de una referencia de un objeto como parámetro, la original y la recibida por parámetro son iguales, es decir, apuntar al mismo objeto; entonces de esta forma cualquier cambio aplicado al objeto se va a ver reflejado una vez

terminado el método llamado.
Todos los objetos wrappers (Byte, Short, Integer, Float, Double, Character, Boolean, String) son inmutables, esto quiere decir que no se puede enviar una copia de la referencia del objeto.
Los métodos en Java pueden devolver un valor con la palabra clave “return”, pueden devolver cualquier tipos primitivo, wrappers y objetos.

Ejemplos:

```
public class Ejemplo {

   public static void funcion1(int a) {
      a = 25;
      System.out.println(a);                  // Imprime 25
   }

    public static void main(String []args){
      int x = 10;
      funcion1(x);
      System.out.println(x);                  // Imprime 10.
    }                                         // Su valor no cambio a pesar que se le estaba sobreescribiendo.
}
```

```
public class Ejemplo {

   public static void funcion2(Character car) {
      car = 'P';
      System.out.println(car.toString());          // Imprime ‘P’
   }

    public static void main(String []args){
      Character c = new Character('A');
      funcion2(c);
      System.out.println(c.toString());            // Imprime ‘A’
    }                                              // Los objetos de tipo wrappers son inmutables.
}
```

```
public class Ejemplo {

   public static String funcion3(String[] s) {
      String temp = s[0] + " " + s[1];
      s[0] = "ABC";
      s[1] = "123";
      System.out.println(s[0]);                // Imprime “ABC”
      System.out.println(s[1]);                // Imprime “123”
      return temp;
   }
    public static void main(String []args){
      String[] s = new String[2];
      s[0] = "Hola";
      s[1] = "Mundo";
      System.out.println(funcion3(s));     // Imprime “Hola Mundo”
      System.out.println(s[0]);                // Imprime “ABC”
      System.out.println(s[1]);                // Imprime “123”
```

```
    }                                   // Muestra en pantalla lo mismo en cada posición porque los
}                                       // objetos (en este caso array[]) envían una copia de su
```
referencia.

**C**

En este lenguaje los parámetros pueden ser enviados de dos formas, por valor y por referencia.

**Por valor**: cuando se envía un parámetro a través de este método el dato de la variable se aloja en una dirección de memoria diferente al recibirla en una función, entonces si la información de la variable cambia no afecta a la variable original. Se hace una copia de la variable original.

**Por referencia:** con este método la variable que se recibe como parámetro apunta exactamente a la misma dirección de memoria que la variable original, entonces si dentro de la función se modifica el valor de la variable el cambio también se ve afectado en la variable original.

Ejemplos:

**- Por valor**

```
#include <stdio.h>

void funcion1(int x) {
    x = x * 0.1;
    printf("valor de x: %d", x);       // Imprime 10
}

int main()
{
    int a = 100;
    funcion1(a);
    printf("valor de a: %d", a);       // Imprime 100
    return 0;
}
```

**- Por referencia**

```
#include <stdio.h>

void funcion2(int *x) {
    *x = *x * 2;
    printf("valor de x: %d", *x);      // Imprime 50
}

int main()
{
    int a = 25;
    funcion2(&a);
    printf("valor de a: %d", a);       // Imprime 50
    return 0;
}
```

## **Matlab**

En este lenguaje las funciones se escriben en un archivo separado y el nombre del archivo debe coincidir con el nombre de la función.
Las funciones pueden aceptar ninguno o más de un argumento de entrada y pueden devolver ninguno o más de un argumento de salida.
La forma de escribir una función es:

function [out1, out2, …, outN] = “nombre” (in1, in2, in3, …, inN)
“instrucciones”

Donde:
- variable: contiene el resultado que devuelve la función.
- nombre: forma de identificar a la función.
- parámetros: son los datos de entrada de la función.
- instrucciones: las sentencias para realizar la tarea de la función.

Algunos ejemplos:

En esta función en ‘x’ se guarda el valor correspondiente al realizarse la condición.

```
function x = menor(a,b)
        if (a<b)
                x = a;
        else
                x = b;
        end
```

Una función puede no devolver ningún resultado.

```
function saludar
        disp(‘Hola’);
```

Una función puede no recibir ningún parámetro.
En ‘total’ se devuelve la suma de los primeros 10 números naturales.

```
function total = sumar
        total = 0;
        for i = 1 : 10
                total = total + i;
        end
```

Una función puede devolver más de un resultado.
En ‘numero1’ y ‘numero2’ se devuelve el dato ingresado por teclado.

```
function [numero1, numero2] = números
        numero1 = input(‘Dato número 1: ’);
        numero2 = input(‘Dato número 2: ’);
```

También se pueden enviar como parámetro o devolver como resultados vectores y matrices.

## **Mathlan**

En nuestro lenguaje se usan los pasajes por valor y por referencia, al igual que en C.

-Por Valor:

- Cuando un argumento es pasado por valor, se hace una copia del valor del argumento y se pasa a la función que es llamada.
- Los cambios a la copia no afectan el valor original de la variable que aparece en la llamada.
- A este tipo de parámetros se les conoce como parámetros de “entrada”.

-Por Referencia:

- Cuando un argumento es pasado por referencia, no se hace una copia del valor del argumento, sino que se está pasando la referencia a la función o subrutina.
- Los cambios a la copia si afectan el valor original de la variable que aparece en la llamada.
- La ventaja del paso por referencia es el menor consumo tiempo y memoria, ya que no se tiene que realizar copias de los datos.