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**
- **https://www.mathworks.com/help/coder/examples/working-with-persistent-variables.html**

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:

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

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:

- 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).
En caso de las variables dinámicas (por ejemplo las variables apuntadas por un puntero) la ligadura se hace en tiempo de ejecución porque es dinamica del l-valor, y la estabilidad depende del momento en el que se hace el dispose de la variable, la estabilidad va a ser entre el new y el dispose, en cambio, en una variable estática la estabilidad va a ser durante toda la vida del programa.
En las constantes el r-valor no cambia durante la ejecución de todo el programa, es decir, que se congela su valor. Por ejemplo, en java a una variable la definimos con un tipo y este tipo se fija en compilación y no va a cambiar durante toda la vida del programa, la estabilidad es estática porque no cambia durante todo el programa.

E) Nuestro lenguaje contará con tres 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 implementa C. 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 solo tienen un modificador de alcance para implementar 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).
Las variables persistentes de matlab, también se las suele llamar variables estáticas, existen desde el momento en que son creadas hasta que se las destruye explícitamente mediante la orden “clear”. Las variables globales son de este tipo.

```
function y = calcular_promedio(x)
    assert(isa(x,'double'));        % Verificamos que x pertenezca a la clase double.
    persistent sum cnt;             % sum y cnt son variables persistentes.
    if isempty(sum)
            sum = 0;
            cnt = 0;
    end
    sum = sum + x;
    cnt = cnt + 1;
    y = sum / cnt;                  % retorno.
```

F) <u>Java</u>

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.

<u>Ejemplos:</u>

```
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;
        función1(x);
        System.out.println(x);              // Imprime 10.
    }                                       // Su valor no cambio a pesar que se le estaba sobreescribiendo.
}
```

```
public class Ejemplo {
      public static void función2(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 función3(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(función3(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[]) envian 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.

CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN

TRABAJO INTEGRADOR 2017

Tercera 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://www.google.com.ar/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=0ahUKEwiE0eXv8qbUAhVJOZAKHYL7AtAQFggnMAE&url=http%3A%2F%2Flaurel.datsi.fi.upm.es%2F~ssoo%2FSTR%2FExcepciones.pdf&usg=AFQjCNENsDDBqjqeO3R3B9N7pjVW7CWPtw&sig2=URlGCGZMEmeNBGOr3C3whA
- https://cvw.cac.cornell.edu/matlab/tips?AspxAutoDetectCookieSupport=1
- http://simplesoftmx.blogspot.com.ar/2014/06/lenguajes-fuertemente-y-debilmente.html
- https://universidad-de-los-andes.gitbooks.io/fundamentos-de-programacion/content/Nivel4/5_ManejoDeLasExcepciones.html
- http://www.it.uc3m.es/abel/as/DSP/M1/CDataTypes_es.html

3ra Parte (Fecha Final de entrega: 9/6)

G. Defina y detalle las características del sistema de Tipos de su lenguaje y compárelo con el de sus lenguajes asignados justificando las decisiones sobre su diseño.

H. Describa el modelo de excepciones elegido para su lenguaje, describa el mecanismo utilizado y compárelo con alguno de los modelos impartidos en la práctica. Compárelo con alguno de los mecanismos de los lenguajes asignados. En todos los casos ejemplifique.

I. Conclusión del trabajo integrador.

G) El uso de tipos de datos puede permitir a un compilador detectar incoherencias en el significado o código probablemente inválido. Por ejemplo, podemos identificar una expresión " 3 / A " como inválida porque no se puede dividir un entero por un caracter. Un sistema de tipado fuerte ofrece más seguridad, pero no garantiza una seguridad completa, si no que intenta lograr que los programas sean lo más seguro posible.
Matlab es débilmente tipado ya que no define de forma explícita el tipo de dato de la variable, sino que el tipo de dato se determina durante la ejecución del programa en función de los valores que se le vayan asignando a la variable. C también es débilmente tipado, ya que a pesar de que se puedan declarar variables indicando su tipo, permite usar variables de un tipo de dato como si fueran de otro tipo de dato diferente, dando como resultado un valor no esperado, por ejemplo:

```
#include <stdio.h>

int main(){
   int a = 4;
   char b = '5';
   float c;
   c = a + b;
   printf("%f", c);          // Imprime 57.00000, porque realizo la suma de 4 + 53 (Código ASCII del caracter
   return 0;                  '5', e imprime el número en decimal porque guardo el resultado en un tipo float.
}
```

Tipos de ligadura:

- **Ligadura estática**: La comprobación del tipo se hace durante el tiempo de compilación. Esto permite que lo errores se encuentren antes (C y Java).

- **Ligadura dinámica**: La comprobación de tipo se hace durante el tiempo de ejecución. Hace que el lenguaje sea más flexible, pero es más propenso a tener errores de programación (Matlab).

En Java y C la ligadura es estática, pero en Java también es dinámica. Esto pasa porque tiene una forma de conversión de tipos de datos forzada, denominada "cast" que se resuelven en tiempo de ejecución.

Tipos de datos de Java:

1. byte
2. short
3. int
4. long
5. float
6. double
7. char
8. boolean
9. Clases definidas por el usuario: el programador puede crear sus propias clases e instanciar.

Tipos de datos en C:

1. int
2. short
3. long
4. long long
5. char
6. char x [SIZE] (tabla para una cadena de caracteres de tamaño SIZE)
7. int matriz [sizeA] [sizeB] (tabla de múltiples dimensiones)
8. float
9. double
10. Tipos de datos definidos por el usuario (por ejemplo, struct y enum)

En C se puede anteponer la palabra unsigned o signed en los tipos de datos numéricos para indicar si es con o sin signo.

Matlab no tiene tipos, todas las variables se comportan como matrices, no necesita ningún tipo de declaración. Ejemplo: x = 50 // Crea una nueva matriz de 1x1 denominada "x" y guarda el valor 50.

Operaciones de C y Java:

+ … **suma …**
- … **resta …**
***** … **multiplicación …**
/ … **división …**
++ … **incremento …**
-- … **decremento …**
% … **modulo …**
> … **mayor que …**
< … **menor que …**
>= … **mayor o igual que …**
<= … **menor o igual que …**
== … **igual que …**
!= … **distinto que …**

! **… negación …**
&& **… AND lógico …**
|| **… OR lógico ...**
= **… asignación …**

Operaciones de Matlab:

+ **… suma ...**
- **… resta ...**
/ **… división ...**
***** **… multiplicación ...**
^ **… potenciación …**
= **… asignación ...**
sqrt() **… raíz cuadrada …**
== **… igual que …**
!= **… distinto que …**
> **… mayor que …**
< **… menor que …**
<= **… menor o igual que …**
>= **… mayor o igual que …**
& **… AND lógico ...**
| **… OR lógico ...**
~ **… NOT lógico ...**

Java es **ortogonal**, es decir, podemos usar mismo operador para varias cosas. Por ejemplo: podemos usar el “+” para concatenar strings y para sumar.

```
public class EjemploDelMas {
    public static void main(String []args) {
        String s1 = "Hola ", s2 = "Mundo. ", s3;
        int x = 10, y = 5, z;
        s3 = s1 + s2;
        z = x + y;
        System.out.println("Resultado de hacer +s1+” + “+s2+ es: " + s3);          // Muestra “Hola Mundo. “
        System.out.println("Resultado de hacer +x+” + “+y+ es: " + z);              // Muestra 15
    }
}
```

Nuestro lenguaje va a ser **fuertemente tipado**, como lo es Java, de este modo se obtienen programas más seguros, pero menos flexibles.
Su ligadura va a ser estática como en C, por lo tanto, cada variable va a quedar ligada a su tipo durante la compilación. Los tipos de datos de nuestro lenguaje van a ser los mismo que Java, y en cuanto a las operaciones tomaremos las de C y Java.

H) Una excepción es la indicación de que se produjo un error en el programa. Las excepciones, como su nombre lo indica, se producen cuando la ejecución de una instrucción no termina correctamente, sino que termina de manera excepcional como consecuencia de una situación no esperada.
Cuando se produce una situación anormal durante la ejecución de un programa (por ejemplo se accede a un objeto que no ha sido inicializado o tratamos de acceder a una posición inválida en un vector), si no manejamos de manera adecuada el error que se produce, el programa va a terminar abruptamente su ejecución.

Existen dos tipos de modelo de manejo de excepciones:

- Reasunción: se maneja la excepción y se devuelve el control al punto siguiente donde se invocó la excepción, permitiendo la continuación de ejecución de la unidad. El lenguaje que lo utilizan es PL/1.

- Terminación: se termina la ejecución de la unidad que alcanza la excepción y se transfiere el control al manejador. Los lenguajes que lo utilizan son ADA, CLU, C++, JAVA.

En Java, las excepciones son objetos que pueden ser alcanzados y manejados por manejadores adicionados al bloque donde se produjo la excepción.

Uso de FINALLY.

Ejemplo:

```
try
    bloque
catch (NombreExcepciónN)
    bloqueManejadorN
finally
    bloqueFinal
```

Finally puede estar o no. Si está, la ejecución de su código se realiza cuando se termina la ejecución del bloque Try, se haya o no levantado una excepción, salvo que el bloque Try haya levantado una excepción que no macheo con ningún manejador.

En C no hay excepciones, las mismas se tendrán que contemplar caso por caso con una estructura condicional.

Ejemplo de excepciones de ADA:

```
with Text_IO;
use Text_IO;
procedure ejemplo is
   a, b, c : integer;
begin
  a:= 10;
  b:= 0;
  c:= a / b;
exception
  when Numeric_Error =>
    new_line;
    put_line("No se puede dividir por cero");
end ejemplo;
```

Nuestro lenguaje va a seguir el modelo de **Terminación**, al igual que Java y Ada, ya que lo que buscamos es que cada vez que se produce una excepción, se termine el bloque donde se levantó dicha excepción y se ejecute el manejador asociado.

Ejemplo de excepciones en Java:

En el siguiente ejemplo se pide la entrada por teclado de 2 datos numéricos para realizar la división entre

ambos números.
La primera instrucción “catch” es para atender la excepción generada por ingresar un dato incorrecto por teclado, sólo se permiten números.
La segunda instrucción “catch” es para atender la excepción generada por intentar dividir un número por 0.

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class Ejemplo {
	public static void main(String[] args) {
		Scanner entrada = new Scanner(System.in);
		int numA, numB, res = 0;
		try {
			System.out.println("Introduce un número: ");
			numA = entrada.nextInt();
			System.out.println("Introduce otro número: ");
			numB = entrada.nextInt();
			res = numA / numB;
			System.out.println("Resultado: " + res);
		} catch (InputMismatchException  input) {
			System.out.println("Error en la entrada. Se ingreso un dato incorrecto. ");
		} catch (RuntimeException errorDivision) {
			System.out.println("Error en la división. No se puede dividir por 0. ");
		}
	}
}
```

I) *Conclusión*:

1. **Características Principales:**
   - ➢ JAVA:
     - **Orientado a objetos**: toda la programación en java en su mayoría está orientada a objeto, ya que al estar agrupados en estructuras en estructuras encapsuladas es más fácil su manipulación.
     - **Portable**: por ser indiferente a la arquitectura sobre la cual está trabajando, esto hace que su portabilidad sea muy eficiente, sus programas son iguales en cualquiera de las plataformas, ya que java especifica tamaños básicos, esto se conoce como la máquina virtual de java.
     - **Seguro**: la seguridad es una característica muy importante en java ya que se han implementado barreras de seguridad en el lenguaje y en el sistema de ejecución de tiempo real.
     - **Simple**: se lo conoce como lenguaje simple porque viene de la misma estructura de c y c++; ya que c++ fue un referente para la creación de java por eso utiliza determinadas características de c++ y se han eliminado otras.
     - **Robusto**: es altamente fiable en comparación con c, se han eliminado muchas características con la aritmética de punteros, proporciona numerosas comprobaciones en compilación y en tiempo de ejecución.
     - **Dinámico**: el lenguaje java es muy dinámico en la fase de enlazado, sus clases solamente actuarán en medida en que sean requeridas o necesitadas con esto permitirá que los enlaces se puedan incluir incluso desde fuentes muy variadas

o desde la red.
- **Compilado**: es un lenguaje que es compilado, generando ficheros de clases compilados, pero estas clases compiladas son en realidad interpretadas por la máquina virtual java. Siendo la máquina virtual de java la que mantiene el control sobre las clases que se estén ejecutando.

➢ C:
- **Portable**: es independiente del hardware. Los programas escritos en C son fácilmente transportables a otros sistemas.
- **Simple**: es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado.
- **Compilado:** convierte el código fuente en un fichero de objeto y este en un fichero o ejecutable.
- **Medio nivel**: combina elementos de lenguaje de alto nivel con la funcionalidad del lenguaje ensamblador.
- **Estructurado**: permite crear procedimientos en bloques dentro de otros procedimientos.

➢ Matlab:
- Cálculos intensivos desde un punto de vista numérico.
- Gráficos y visualización avanzada.
- Lenguaje de alto nivel basado en vectores, arrays y matrices.
- Colección muy útil de funciones de aplicación.

2. **Ventajas y Desventajas**:
➢ JAVA:
○ **Ventajas**:
- La principal característica de Java es que es independiente de la plataforma (multiplataforma). Esto significa que cuando estás programando en Java, no necesitas conocer a priori el tipo de ordenador o el sistema operativo para el que estás programando. Puedes ejecutar EL MISMO programa en un PC con Windows, otro con Linux, en un Servidor SUN con sistema operativo Solaris, o en un teléfono móvil de última generación.
- El lenguaje Java es orientado a objetos. El paradigma de programación orientada a objetos supuso un gran avance en el desarrollo de aplicaciones, ya que es capaz de acercar la forma de programar a la forma de pensar del ser humano.
- En java no existen problemas con la liberación de memoria en el sistema: En Java decidieron romper con el sistema tradicional de liberación de memoria, haciendo que el programador ya no fuese el responsable de esa tarea. Así, lo único que necesita hacer el programador es solicitar la memoria al sistema.
- El lenguaje Java es relativamente fácil de aprender comparado con otros.
- Librerías Estándar: Una de las características que más potencia aporta al lenguaje Java es que viene acompañado de una serie de librerías estándar para realizar multitud de operaciones comunes a la hora de programar. Es el llamado Java API, que incluye tres bloques básicos.
- Hoy en día existen excelentes editores (IDEs) que aportan multitud de ayudas a la programación, haciendo que el desarrollo sea más fluido y cómodo.
- Una de las soluciones más elegantes propuestas por el lenguaje Java a uno de los problemas recurrentes en otros lenguajes de programación es la gestión de errores a través de excepciones. en C o C++ no existe un

mecanismo específico para la gestión de los errores que puedan producirse en el código.

- **Desventajas**:
  - Menos Eficiente, comparado a C/C++.
  - Requiere un intérprete.
  - Algunas implementaciones y librerías pueden tener código rebuscado.
  - Una mala implementación de un programa en java, puede resultar en algo muy lento.
  - Algunas herramientas tienen un costo adicional

➢ C:

- **Ventajas**:
  - Es un lenguaje potente y eficiente, permitiendo obtener programas rápidos y compactos.
  - Proporciona un completo control de cuanto sucede en el interior del ordenador.
  - Permite una amplia libertad de organización del trabajo.

- **Desventajas**:
  - Es más complicado de aprender que otros lenguajes de programación como por ejemplo el Pascal.
  - Requiere una cierta experiencia para poder sacarle el máximo rendimiento.
  - Sin disciplina es difícil mantener el control del programa.

➢ Matlab:

- **Ventajas**:
  - Presenta grandes ventajas a la hora de trabajar con números complejos, matrices con polinomios, funciones trigonométricas, logaritmos, simulación de sistemas dinámicos, visión artificial, análisis estadístico, análisis y diseño de controladores automáticos, etc.

- **Desventajas**:
  - Como principal inconveniente hay que señalar el hecho de que matlab no ha sido concebido como lenguaje de programación, por lo que carece de algunos elementos o características necesarias para una buena práctica de la programación.

Comparación entre C y JAVA:

- Java es programación orientada a objetos, a diferencia de C que es de programación estructurada.
- C es mucho más rápido.
- C es un lenguaje rápido y compilado, lo contrario de los que son interpretados. Claro eso sí que al agregar rapidez, agregamos también dificultad en su uso. Y Java es un lenguaje, medio compilado, medio interpretado. A qué me refiero, me refiero a que Java necesita una compilación previa para poder ejecutarse en su intérprete: JVM (*Java Virtual Machine*: Máquina Virtual de Java).
- Java es más seguro. Cualquier indicio de inseguridad (referencias mal hechas, operaciones peligrosas, etc.) en tu programa se traducirá en un error. La potencia de C es

la que le da su peligrosidad, el acceso libre a la memoria, y otros factores que lo hacen el lenguaje más potente después de Assembler.
- Java es un poco menos complicado para los novatos por su no uso de punteros. Los punteros (o referencias) se deben utilizar con cuidado (en C) ya que dan un acceso ilimitado a lo que es la memoria y disco duro del computador. Es por ello la capacidad de crear código malicioso con este lenguaje.
- C es en realidad un lenguaje de bajo nivel (más entendible para la máquina) con instrucciones de alto nivel (más comprensible por humanos), o sea, no muy complicado de aprender para que sea de bajo nivel. En cambio, Java posee pocas instrucciones de bajo nivel, con poco acceso al sistema y menor gestión de memoria que con C, o sea, más fácil pero menos control.

Comparación entre Matlab y Java:

- Matlab tiene mucho más apoyo a las operaciones matemáticas de alto nivel, al igual que la multiplicación de matrices. Se puede escribir (o encontrar) bibliotecas para hacer estas operaciones en Java, pero es mucho más trabajo.
- Matlab es interpretado, no compilado como Java. Esto hace que sea fácil de experimentar de forma interactiva.
- Matlab funciona más lentamente que Java, excepto para hacer operaciones con matrices incorporadas como encontrar los valores (para los que MATLAB es generalmente más rápido).
- Matlab es caro, mientras que se puede descargar de forma gratuita de Java.

Nuestro lenguaje Mathlan:

Las características más relevantes son que posee una gran variedad de tipos de datos primitivos, mas la opcion de que el programador pueda definir sus propias clases para poder instanciar y usar esos objetos para resolver cualquier problema, además de contar con un gran número de librerías.
Terminamos tomando más características del lenguaje Java porque es el lenguaje más completo, seguro y simple respecto de C y Matlab.

Nuestra conclusión final sobre el trabajo:

La cursada es bastante diferente a las materias que venimos cursando años anteriores, hay que realizar mucha investigación bibliográficas y a través internet. Nos resultó interesante ver diferentes tipos de lenguajes de programación y las distintas características que posee cada uno, de algunos pudimos ver mas a fondo cómo es su sintaxis y semántica, como por ejemplo los lenguajes de C, Matlab y Ada.
Con el tema del trabajo, nos costó entender gran parte de las consignas, ya que el trabajo está poco claro y perdimos tiempo en consultarlas y entenderlas, pero pudimos llevarlas bien las dos primeras partes del trabajo y en esta última entrega nos costó un poco más.