Trabajo Integrador

Primera Etapa

Grupo 43

Integrantes

- 15980/7 Molina, Lucio Felipe.
- 16286/5 Ajenjo, Tobías Adrián.
- 16372/2 Ramírez, Sergio Daniel.
- 16440/6 Pereyra, Iyael Lihue.

Lenguajes Asignados

- Lenguaje C.
- Lenguaje Java.

Bibliografía Utilizada

- Kerninghan, Ritchie. (1991). El Lenguaje de Programación C. Pearson.
- Mala Gupta. (2017) OCA Java SE 8 Programmer I Certification Guide.
  Manning Publications.
- Pablo Sznajdleder. (2015). Java a Fondo – Curso de Programación.
  Alfaomega.

Enunciado

A. Enuncie y compare distintas características (o criterios de evaluación) de cada uno de los lenguajes asignados fundamentando cada una con ejemplos de código.

B. Defina y compare diferentes aspectos de la sintaxis que Ud. considere. Ejemplifique.

C. Enuncie y compare distintos aspectos semánticos (tanto de la semántica estática y dinámica) de cada uno de los lenguajes asignados.

D. Defina una porción de código donde pueda apreciarse las características más relevantes de las variables en cuanto a sus atributos. Elija alguno de los lenguajes asignados que presente mayores posibilidades para mostrar estas características y desarrolle el ejercicio de la misma forma que se realiza en la práctica. Luego, si es necesario realice las explicaciones que permitan una mayor comprensión del ejercicio.

A.

Simplicidad y legibilidad

Mas allá de que el Lenguaje C posea varias librerías para realizar determinadas funciones, el código Java ya posee implementado determinados mensajes y resulta más simple a la hora de hacer un print por ejemplo.
En el caso de C, se necesitaría la librería #includes<stdio.h>.

Lenguaje C

```
1. #includes<stdio.h>
2. int main (){
3.     printf(“Hola mundo”)
4.     return 0;
5. }
```

Lenguaje Java

```
1. public class HolaMundo1{
2.     public static void main(String[] args){
3.         System.out.print(“Hola mundo”);
4.     }
5. }
```

Abstracción

Si bien ambos lenguajes poseen diversos métodos para abstraerse de los detalles de más bajo nivel, Java al ser un lenguaje **Orientado a Objetos** posee una mayor abstracción que C, ya que permite definir datos y lógica de forma conjunta a través de objetos, además de proveer diversas mecánicas, como lo es la **herencia** o **polimorfismo**, que permiten al programador abstraerse de procesos complicados y verlo de una manera simple.

Seguridad

En el caso de Java, si el **bytecode** pasa la verificación sin generar ningún mensaje de error, significa que nuestro código es seguro y no intentan acceder a datos privados.

En cambio, C tiene lagunas de seguridad importantes, como son los errores de **alineación**. Los programadores de C utilizan punteros en conjunción con operaciones aritméticas. Esto le permite al programador que un puntero referencie a un lugar conocido de la memoria y pueda sumar (o restar) algún valor, para referirse a otro lugar de la memoria.

- Pedir memoria: **malloc()**
- Liberar memoria: **free**()

Lenguaje C

```
1.  #include <stdlib.h>

2.  int* ptr;        /* puntero a enteros */
3.  int* ptr2;       /* otro puntero */

4.  ...

5.  /* reserva hueco para 300 enteros */
6.  ptr = (int*)malloc ( 300*sizeof(int) );
7.  ...
8.  ptr[33] = 15;           /* trabaja con el área de memoria */

9.  rellena_de_ceros (10,ptr);  /* otro ejemplo */

10. ptr2 = ptr + 15;                /* asignación a otro puntero */

11. free(ptr);       /* finalmente, libera la zona de memoria */
```

Eficiencia

Ambos lenguajes son muy eficientes ya que C es el lenguaje que mejor aprovecha la CPU de la máquina, sin tener las desventajas de los lenguajes ensambladores.
Sin embargo, Java es uno de los lenguajes de programación orientado a objetos **más rápidos y más eficientes en el uso de energía**.
La desventaja de C es que no posee **Garbage Collector**. En lenguajes como Java, la liberación de la memoria es automática, haciendo que de vez en cuando, elimine lo que no se está usando. En C el control de la memoria lo tiene uno, así que hay que tener claro cuando liberarla.

Soporte

El lenguaje C no posee tanta **portabilidad**, ya que se puede tener problemas y no funcionaría en todos los sistemas.
El código fuente es portátil, pero se debe generar un nuevo binario para cada sistema operativo.
Al ser un lenguaje antiguo no se encuentra tanta documentación como Java.
La **JVM** de Java es una máquina virtual de proceso nativo que se puede ejecutar en una plataforma específica para el que fue diseñado, capaz de interpretar el bytecode Java.
El lenguaje Java es un lenguaje fácil de aprender y muy utilizado, posee mucha documentación y soporte.

Ortogonalidad

Java es ortogonal, ya que hay un conjunto relativamente pequeño de construcciones primitivas que se pueden combinar en un número relativamente pequeño de maneras de construir estructuras de datos y de control.
En C, en cambio, uno de los ejemplos más relevantes es que no se puede devolver un array, perdiendo la así este principio.

Ejemplo de funciones con retorno de arrays en Java y C

Lenguaje Java

```
1.  public static int[] numbers() {
2.      int[] arr = {5, 6, 7, 8,9};
3.      return arr;
4.  }
5.  public static void main(String args[]) {
6.      int[] a = numbers();
7.      for (int i = 0; i < a.length; i++)
8.      System.out.print(a[i] + " ");
9.  }
```

Lenguaje C

```
1.  #include <stdio.h>
2.  int getarray()
3.  {
4.      int arr[5]={25,63,14,22,20};
5.      return arr;
6.  }
7.  int main()
8.  {
9.      getarray();
10.     return 0;
11. }
```

Este código genera un error en tiempo de compilación.

B.

Ejemplo de palíndromo

Utilizamos un ejemplo de código para diferenciar los distintos aspectos de la sintaxis, implementado en Java y C.

Lenguaje C

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #define TAM 100
4.  int Palindroma(char *cadena) {
5.      register int i,j;
6.      i=0;
7.      j=strlen(cadena)-1;
8.      while (i<j && cadena[i]==cadena[j]) {
9.          i++; j--;
10.     }
11.     return (i>=j);
12. }
13. int main(void) {
14.     char cadena[TAM];
15.     printf("\nIntroduce la palabra\n");
16.     gets(cadena);
17.     printf("La palabra: %s %s palindroma.\n",cadena, (Palindroma(cadena)) ?
    "es" : "no es");
18.     return 0;
19. }
```

Con respecto a las declaraciones de variables y bibliotecas

- Para imprimir en la consola se utiliza la biblioteca include<stdio.h>.
- Cuando se escribe **TAM 100** se define un alias identificado por “TAM” a un valor integer 100. Se utiliza para declarar el array de char.
- string.h es un archivo de la biblioteca estándar del lenguaje de programación C que contiene la definición de macros, constantes, funciones y tipos y algunas operaciones de manipulación de memoria.

Diferencias respecto a la sintaxis concreta

- gets hace referencia al scanf, lo que hará es que se pida un valor por teclado.
  En Java se utiliza Scanner.
- En los printf, antes de imprimir el código, hay que mencionar de qué tipo de dato se va a imprimir por consola como el %s (imprime string).
  En Java no es necesario indicar su tipo.
- Los booleanos al ser integers (true=1 y false=0), cuando se evalúa la función en el string verifica si el resultado de la función es true o false.
  En el caso de Java, los valores del boolean son true y false.
- strlen(cadena)-1 nos dará el tamaño de la palabra. Se resta uno para referenciar al último carácter.
- En el printf se hace una condición de si da true imprime “es” sino se imprimirá “no es”, esto se hace con la operación terciaria “?”.
- Al final del main se retorna un 0 que significa la finalización del programa.

El ejemplo incluye las **palabras reservadas** char, return y while.

Lenguaje Java

```
1.  import java.util.*;
2.  class PalindromeExample
3.  {
4.     public static void main(String args[])
5.     {
6.        String original, reverse = ""; // Objects of String class
7.        Scanner in = new Scanner(System.in);
8.        System.out.println("Enter a string/number to check if it is a palindrome");
9.        original = in.nextLine();
10.       int length = original.length();
11.       for ( int i = length - 1; i >= 0; i-- )
12.          reverse = reverse + original.charAt(i);
13.       if (original.equals(reverse))
14.          System.out.println("Entered string/number is a palindrome.");
15.       else
16.          System.out.println("Entered string/number isn't a palindrome.");
17.    }

18. }
```

Inicialmente podemos ver la definición de una función

- La utilización de modificadores de acceso que define su alcance.
- La palabra reservada "**static**" que define su tiempo de vida. En este caso, al ser static, se define un espacio de nombres para la función momentos antes de su ejecución.
- El tipo de retorno "**void**".
- Podemos ver el identificador "**main**", que debe respetar las reglas definidas para la creación de identificadores.
- La definición de parámetros, como variables locales, su identificador y tipo.

Adicionalmente podemos ver la definición de variables de **tipos primitivos**, como es el caso de la variable "length" de tipo int. La variable "in" de tipo Scanner, es un ejemplo de uso de tipos definidos por el usuario.

También el uso de **estructuras de control de flujo**. En ambos lenguajes tanto el bucle for, como el while y el condicional if, si bien a nivel de sintaxis concreta pueden diferir, son iguales respecto a la **sintaxis abstracta**.

En el bucle for se puede ver la definición de la variable "i", que tiene **alcance** y **vida** en el contexto del bucle.

La utilización de **keywords**, como es el caso del modificador de acceso "public", el bucle "for", el condicional "if", el tipo de retorno "void", entre otras.

Respecto a la sintáxis abstracta

Ambos lenguajes tienen similitudes en las estructuras de control.

```
for (inicializador, hasta donde, incremento)
    bloque

while(condicion)
    bloque

if(condicion)
    bloque
```

En la parte de importación de librerías también son similares, solo que en java la sentencia es **import** y en C la sentencia es **include**.

C.

Semántica Dinámica

Ejemplo de cómo Java cachea a los objetos String literales.

Lenguaje C

```
1. int main(void) {

2.     char saludo1[] = "hola";
3.     char saludo2[] = "hola";

4.     printf("Los contenidos son iguales: %s\n", strcmp(saludo1, saludo2) == 0 ?
   "true" : "false"); //strcmp retorna 0 si matchean

5.     printf("Las direcciones son iguales: %s\n", saludo1 == saludo2 ? "true" :
   "false");

6.     return 0;

7. }
```

```
Output:
        Los contenidos son iguales: true
        Las direcciones son iguales: false
```

Lenguaje Java

```
1. public static void main(String[] args) {

2.     String saludo1 = "hola";
3.     String saludo2 = "hola";

4.     System.out.println("Los contenidos son iguales: " +
   saludo1.equals(saludo2));
5.     System.out.println("Las direcciones son iguales: " + (saludo1 == saludo2));

6. }
```

```
Output:
        Los contenidos son iguales: true
        Las direcciones son iguales: true
```

Como se puede ver en el ejemplo, en Java cuando se inicializa una variable de tipo String con un literal, este objeto se crea y se almacena en un **pool de objetos String**. En una próxima inicialización, antes de crear un nuevo objeto en el pool, Java busca un objeto con contenido similar, y si este existe retorna su **referencia**.

Otro ejemplo en donde trabaja de forma diferente la semántica entre los lenguajes C y Java.

Lenguaje C

```
1. int get_number() {
2.     int x;
3.     return x + 1;
4. }
```

Lenguaje Java

```
1. public int get_number() {
2.     int x;
3.     return x + 1;
4. }
```

Como se puede notar, la estructura (forma) del código es idéntica, la diferencia que hay es que, en el ejemplo en C, la función te devuelve un número cualquiera, en cambio en el ejemplo de Java, **el código lanza un error**. La diferencia está en que una variable de tipo integer definida en una función en C se inicializa de manera por defecto con **información basura** que se encuentra en memoria. En cambio, Java lanza un error porque **no inicializa la variable**, en consecuencia, luego lanza un error debido a que se intenta sumar esta variable con el valor 1.

Ejemplo de inferencia de tipos y sus posibilidades entre los lenguajes Java y C.

Lenguaje Java

```
1.  public static void main(String[] args) {
2.          String message1 = "Saludos";
3.          System.out.println(message1);
4.          var message2 = "Saludos con inferencia de tipos!";
5.          System.out.println(message2);
6.  }
```

```
Output:
        Saludos
        Saludos con inferencia de tipos!
```

Lenguaje C

```
1.  int main(void) {
2.     char message1[] = "Saludos";
3.     printf("%s\n", message1);
4.     message2 = "Saludos con inferencia de tipos!";
5.     printf("%s\n", message1);
6.     return 0;
7.  }
```

```
Output:
        Error
```

Como se puede ver en el ejemplo, Java permite la **inferencia de tipos** desde su versión 10. Se debe declarar la variable utilizando la palabra reservada '**var'** e inmediatamente inicializarla. En el caso de C no permite la inferencia de tipos para definición de variables. El código falla en la línea 4 al intentar declarar la variable 'message2' sin definir su tipo.

D.

Ejemplo de código escrito en el lenguaje C:

```
1.  const char welcome_message[] = "welcome!".
2.
3.  void get_multiples(const int number, const int quantity) {
4.      int *array;
5.      static int number_of_times_invoked = 0;
6.
7.      number_of_times_invoked++;
8.      array = malloc(quantity* sizeof(int));
9.
10.     for (int i = 0; i < quantity; i++)
11.         array[i] = number *(i + 1);
12.
13.     {
14.         char array[quantity];
15.
16.       for (int i = 0; i < quantity; i++)
17.           array[i] = 0;
18.     }
19.
20.     for (int i = 0; i < quantity; i++)
21.         printf("%d\n", array[i]);
22.
23.     printf("this function was invoked %d times\n", number_of_times_invoked);
24.
25.     free(array);
26. }
```

Aclaración: Para el alcance de los identificadores se consideró desde la línea de su declaración.

Nombre	Tipo	Alcance	Life Time	Left-Value	Right-Value
welcome_message	char[]	1 - 26	1 - 26	automática	“welcome!”
number	int	3 - 26	3 - 26	automática	indefinido
quantity	int	3 - 26	3 - 26	automática	indefinido
array	int*	4 - 14, 18 - 26	3 - 26	automática	basura
number_of_times_invoked	int	5 - 26	<1 - 26>	estática	0
variable anónima apuntada por “array”	int[]	8 - 14, 18 - 25	8 - 25	dinámica	basura
i	int	10 - 11	10 - 11	automática	0
array’	char[]	14 - 18	13 - 18	semi-dinámica	basura
i’	int	16 - 17	16 - 17	automática	0
i’’	int	20 - 21	20 - 21	automática	0

En este ejemplo de código escrito en el lenguaje C se pueden observar diversos aspectos de los **atributos** de las variables.

En cuanto al alcance de las variables, se puede observar como la mayoría son de alcance **local** al contexto donde fueron declaradas (salvo “welcome_message” la cual tiene alcance **global**). Se puede notar como la variable “array”, definida en la línea 4, pierde alcance al ser redefinida en un contexto más interno al que fue declarada (se redefine dentro del bloque definido entre las líneas 13 y 18).

En cuanto al atributo left-value, se pueden observar las 4 variantes (estática, semi-estática, semi-dinámica y dinámica). La variable “number_of_times_invoked” al ser de tipo **estática** (referido al left-value), persistirá en memoria durante toda la ejecución del programa. En cambio, las demás variables serán alocadas y liberadas de memoria junto a sus respectivas unidades o contextos.

El ejemplo además muestra variables constantes y no constantes, referido al atributo right-value de las variables.

Trabajo Integrador

Segunda Etapa

Enunciado

E. Mencione y compare entre ambos lenguajes los diferentes tipos de parámetros y características de su implementación.

F. Analice y explique la fortaleza del sistema de tipos de cada uno de los lenguajes asignados.

G. Enuncie las características más importantes del manejo de excepciones que presentan los lenguajes asignados.

H. Conclusión sobre los lenguajes: Realice una entrevista a un usuario/s experimentado de los lenguajes asignados con el fin de obtener una opinión acerca del lenguaje respecto de los temas tratados en el trabajo. Esta conclusión no debe superar una carilla.

I. Conclusión sobre el trabajo: Realice una conclusión mencionando los aportes que le generó la realización del trabajo en comparación con sus conocimientos previos de los lenguajes asignados.

E.

Parámetros en Java

En Java todos los parámetros de los métodos se pasan por valor. Cuando se realiza la llamada a un método, los parámetros formales (parámetros indicados en la declaración del método) reservan un espacio en memoria independiente y reciben los valores de los parámetros reales (parámetros indicados en la llamada al método).

Cuando el argumento es de tipo primitivo, el paso por valor significa que durante la ejecución del método se reserva un nuevo espacio para el parámetro formal y no se puede modificar el parámetro real durante la ejecución del método.
Cuando el argumento es de tipo referencia (por ejemplo, un array, un objeto, etc.) el paso por valor significa que no puede modificarse la referencia, pero se pueden realizar llamadas a los métodos del objeto y modificar el valor asignado a las variables miembro accesibles del objeto durante la ejecución del método.

Java también permite el pasaje de parámetros por valor constante por medio de la palabra reservada “final”, y el pasaje de parámetros por resultado de función por medio de la palabra reservada “return”.

Parámetros en C

El lenguaje C, sólo permite pasar parámetros por valor. Si se desea que los cambios efectuados en una función sobre una variable afecten fuera del alcance de la función, es posible simular un paso por referencia mediante el uso de punteros. Si a una función le pasamos como argumento la dirección de memoria de una variable, cualquier modificación que se realice en esa dirección, afectará, lógicamente, al valor que tiene la variable original, y con ello, conseguiremos el mismo efecto que un paso por referencia.

C también permite el pasaje de parámetros por valor constante por medio de la palabra reservada “const”, y el pasaje de parámetros por resultado de función por medio de la palabra reservada “return”.
En C no se pueden devolver vectores como resultado de una función.

F.

Sistema de tipos en Java

Java es un lenguaje fuertemente tipado y tipado estático.
Una de sus **fortalezas** es que éste provee maneras de trabajar entre diferentes tipos de datos **primitivos** de manera **segura**.

Para esto proporciona los siguientes mecanismos.

Reglas de Equivalencia y Conversión

- Los tipos de datos numéricos son compatibles entre sí.
- Char y boolean no son compatibles entre sí

Formas de conversión

- **Conversión automática de tipos:**
  Solo se da cuando los tipos de datos son **compatibles** y cuando se asigna el valor de un tipo de dato más pequeño a uno más grande.

  Ejemplo:

```
public static void main(String[] args)
{
    int i = 100;

    //conversion
    long l = i;

    //conversion
    float f = l;

    System.out.println("Valor Int "+i);
    System.out.println("Valor Long "+l);
    System.out.println("Valor Float "+f);
}
```

```
Output:
        Valor Int 100
        Valor Long 100
        Valor Float 100.0
```

- **Conversión explícita de tipo o casting:**
  Se usa para asignar un valor de tipo de dato más grande a un tipo de dato más pequeño.

  Ejemplo:

```
public static void main(String[] args)
{
   double d = 100.04;
   //casting
   long l = (long) d;
   //casting
   int i = (int) l;
   System.out.println("Valor Double "+d);
   //parte fraccionaria perdida
   System.out.println("Valor Long "+l);
   //parte fraccionaria perdida
   System.out.println("Valor Int "+i);
}
```

```
Output:
        Valor Double 100.04
        Valor Long 100
        Valor Int 100
```

- **Casting por valor fuera de rango en Expresiones:**
  Java automáticamente promueve cada operando byte, short, o char al evaluar una expresión.
  Si un operando es long, float o double, la expresión se promueve a long, float o double, respectivamente.

  Ejemplo:

```
public static void main(String args[])
{
  byte a = 40;
  byte b = 50;
  byte c = 100;
  int d = a * b / c;
}
```

  En este caso, el resultado de a*b **sobrepasa** el **rango** del tipo byte. Para solucionarlo, Java **promueve** cada operando byte o short a int.

- **Casting de Tipo explícito en Expresiones:**
  Al evaluar expresiones, el resultado se actualiza **automáticamente** a un tipo de datos más grande del operando. Pero si almacenamos ese resultado en un tipo de datos más pequeño, genera un **error en tiempo de compilación**, por lo que debemos "**castear**" el resultado.

  Ejemplo:

```
public static void main(String args[])
{
    byte b = 50;
    //casting de tipo int a byte
    //mostraría error si no detallamos (byte)
    b = (byte)(b * 2);
    System.out.println(b);
}
```

```
Output:
        100
```

- **Tiempo de chequeo:** "Java usa tipado estático 'no puro'".

  Java realiza **chequeo de tipos durante la compilación**. En una asignación entre punteros el compilador verifica que los tipos sean compatibles.

  Además, Java realiza **chequeo de tipos durante la ejecución**. Cuando un programa usa un **cast** para accesar un objeto como si fuese de un tipo específico, se verifica durante la ejecución que el objeto en cuestión sea compatible con el cast que se le aplica. Si el objeto no es **compatible**, entonces se levanta una excepción que informa al programador la línea exacta en donde está la fuente del error.

  Esto, junto a que es un lenguaje fuertemente tipado, lo vuelve sumamente "robusto".

Finalmente, su fortaleza más grande es: **La herencia**

- **Java como lenguaje polimórfico:**
  Al ser un lenguaje **orientado a objetos**, éste provee herramientas para manejarlos. La **herencia**, le da flexibilidad, legibilidad y **abstracción**, facilitando la tarea del programador.

  La capacidad de una **clase** u **objeto** de "entender" un mensaje definido en una o más clases con una lógica jerárquica, permite **no repetir** grandes cantidades de código y facilita la abstracción, **modularización** de un programa y el agregado nuevas funcionalidades.

Sistema de tipos en C

C es un lenguaje "**medianamente**" tipado y tipado **estático**.

El término "medianamente" tipado, se refiere a que más allá que en C se debe aclarar el tipo al declarar una variable, se puede dar que, mediante un casteo, errores de tipo pueden que pasen inadvertidos sin dar error, lo que dificulta su depuración.

C permite manejo de operaciones con diferentes tipos de datos. Tiene las formas de casteo denominados **Casting de Tipo explícito** y **Conversión automática de tipos**, que nombramos en el sistema de tipos de Java. Estas funcionan de la misma manera, ya que Java está en parte basado en C. Con el agregado que en C hay modificadores de tipo que vuelven un poco más complejo su manejo, pero permiten más acciones.

Su principal fortaleza son los modificadores de **acceso** y especificadores de **almacenamiento**, que permiten el manejo seguro de un dato a través de las funciones de un programa e incluso del exterior.

G.

Excepciones en Java

Modelo por **terminación**. La unidad que generó la excepción busca manejar la excepción y termina.

Las excepciones que maneja el lenguaje son clases, las cuales extienden la clase **Exception**. Se instancian como cualquier clase, mediante el constructor **new**.

Existen una serie de excepciones ya definidas por el lenguaje (**built-in exceptions**, por ejemplo, **ClassNotFoundException**, **IOException**, etc), y se permite al usuario crear nuevas excepciones (**User defined exceptions**).

Posee propagación dinámica (**stack trace**): una vez lanzadas, en caso de no ser tratadas por la unidad que las generó, ***se propagan dinámicamente***. Si un método puede generar excepciones, pero decide no manejarlas localmente, debe enunciarlas en su encabezado (**throws**).

Los bloques de código que manejan excepciones se distinguen por las palabras clave **try** y **catch.**

Las excepciones se pueden **lanzar** explícitamente (**throw**), o se alcanzan por una condición de error (en el caso de las provistas por el lenguaje).

Consideraciones:
- Puede haber tantos catch como excepciones maneje el método.
- Se pueden anidar los bloques try/catch.
- Si se omite el bloque catch, pero hay un bloque finally, este último se ejecuta antes de propagar la excepción en forma dinámica.

Excepciones en C

El lenguaje C no provee un mecanismo de manejo de excepciones.

H.

La entrevista hecha al profesor Fernando López nos clarificó varios puntos sobre la importancia del lenguaje C.

Tratamos temas como los escenarios donde se desempeña el lenguaje. El **desarrollo de sistemas operativos** es uno de ellos, utilizado para el desarrollo de Kernels de Linux/*BSD y Windows. Además, en el mundo del cálculo también tuvo un rol importante, como es el desarrollo de bibliotecas como **GMP**. También herramientas que al día de hoy son imprescindibles, como es el caso del desarrollo de interfaces de usuario, mediante la biblioteca de componentes gráficos **GTK**. El desarrollo de intérpretes y compiladores de lenguajes son otros ejemplos.

Otro tema tratado fue el incentivo de aprender el lenguaje para los estudiantes. Nos habló del rol del lenguaje en temas como **microcontroladores para IoT**, que en la actualidad sigue creciendo día a día y se da por sentado que será una realidad. También, sobre la conveniencia en varias materias como Sistemas Operativos, Sistemas Paralelos.

Sin dudas, fue tal la importancia del lenguaje C como C++ en el desarrollo de librerías, que al día de hoy los lenguajes más modernos se siguen apoyando en estas. Un ejemplo es Java, que define un keyword (**native**) para el acceso a funcionalidades implementadas en estos lenguajes. Varios frameworks de desarrollo de videojuegos de Java, como es el caso de **LWJGL** o **LibGDX**, se apoyan en librerías desarrolladas en C y C++.

I.

El haber trabajado en profundidad con estos lenguajes nos **amplió el conocimiento** que teníamos de ellos. Ciertos aspectos que uno a veces no tiene en cuenta, o supone que funciona de determinada forma, pudimos conocer bien cómo funcionan y entenderlos.

Además, el trabajo nos proporcionó varias herramientas que tendremos en cuenta al aprender nuevos lenguajes en el futuro. Estas herramientas nos proveen un **enfoque distinto** para abordar con mayor facilidad la complejidad de un lenguaje de programación. El tener un **sólido conocimiento** sobre las características del lenguaje y cómo funciona, y no sólo saber implementar cierta funcionalidad, nos hace mucho más competentes para resolver problemas complejos.

Otro aporte que nos dejó el trabajo es la **dinámica de aprendizaje basada en la comparación de lenguajes**. Presentar las características de forma contrapuesta nos permite analizar las distintas implementaciones y sus ventajas y desventajas. Como en el caso de estos lenguajes, también podemos comparar épocas. Ver como los lenguajes fueron abstrayendo su complejidad más y más, muchas veces en detrimento de su performance.