+++ date = '2025-02-21T10:20:40-08:00' draft = false title = 'Practica 1: Elementos básicos de los lenguajes de programación' +++

Biblioteca.c

*El programa que analizaremos es un sistema para una biblioteca programado en **Lenguaje C**,el primer codigo que analizaremos en esta practica sera el de **biblioteca.c** el cual es el programa principal de este sistema de bibliotecas*

Nombres

*Los nombres se utilizan para identificar variables, funciones, tipos de datos, estructuras, etc. En este código, ejemplos de nombres son **library**, **members**, **book_t**, **genre_t**, **addBook**, **findBookById**, entre otros.*

Objetos

*Los objetos son instancias de estructuras, como **book_t** y **member_t**. Son contenedores de datos que agrupan atributos relacionados. Por ejemplo, un libro está representado por un objeto de tipo **book_t**.*

Entornos

Un entorno se refiere al contexto en el que se evalúan las expresiones. En este código, el entorno sería el ámbito en el que las variables están definidas, como las variables globales o las variables locales dentro de cada función.

Bloques

*Los bloques se delimitan con **{}**. Cada función está contenida en un bloque de código. Por ejemplo:*

```
void addBook(book_t **library, int* count) {
    // Bloque de código de la función
}
```

Alcance

El alcance se refiere a la visibilidad y duración de las variables. En este caso:

- *Variables locales como **bookFound** y **memberFound** son visibles solo dentro de la función en la que se definen.*
- *Variables globales como **library** y **members** son accesibles en todo el programa.*

Administración de memoria

*La administración de memoria está relacionada con la asignación y liberación de memoria. El código usa **malloc** para asignar memoria dinámicamente y **free** para liberar memoria cuando ya no es necesaria. También*

se manejan las reasignaciones con realloc para ajustar el tamaño de los arreglos, como en el caso de los libros prestados a un miembro.

Expresiones

Las expresiones son combinaciones de valores, variables, operadores y funciones que producen un resultado. Ejemplo de una expresión en el código:

```
current->quantity--;
```

Comandos

*Los comandos en el código incluyen las instrucciones que el programa sigue, como **printf()**, **scanf()**, **malloc()**, **free()**, etc. Estas son las operaciones que ejecutan acciones en el sistema.*

Secuencia

La secuencia es la ejecución de instrucciones de manera lineal, una después de otra. En el programa, por ejemplo, las acciones que el usuario selecciona en el menú se ejecutan de manera secuencial según la opción que elija el usuario.

Selección

La selección se refiere a la ejecución de un bloque de código dependiendo de una condición. Ejemplo de selección:

```
if(bookFound && memberFound)
{
    // Código si la condición es verdadera
}
else
{
    // Código si la condición es falsa
}
```

Recursión

La recursión ocurre cuando una función se llama a sí misma. En este código, la función displayBooksRecursive es recursiva:*

```
void displayBooksRecursive(book_t *library) {
    if (!library) {
        return;
    }
    // Llamada recursiva
    displayBooksRecursive(library->next);
```

```
}
```

Subprogramas

*Los subprogramas son bloques de código que realizan tareas específicas y se invocan desde otras partes del programa. Las funciones como **addBook()**, **issueBook()**, y **displayBooks()** son ejemplos de subprogramas en el código.*

Tipos de datos

*Los tipos de datos definen qué tipo de valores pueden almacenar las variables. En el código, se utilizan tipos como **int**, **char[]**, **enum** y **struct**.*

Memory_managment.c

*Ahora continuaremos con otro programa el cual es **Memory_Managment.c** lo desglosaremos de la misma manera que el otro*

Nombres

*Los identificadores como **heap_allocations**, **heap_deallocations**, **stack_allocations**, **stack_deallocations**, **MemoryRecord**, entre otros.*

Objetos

*En este caso, la estructura **Memory_Record**, es un objeto que guarda información sobre las asignaciones de memorias dinamicas. Cada nodo de esta estructura almacena un puntero y el tamaño de la memoria asignada.*

Entornos

*lo que incluye la memoria (heap y stack). El código gestiona explícitamente el entorno de memoria al contar las asignaciones y liberaciones de memoria, usando contadores como **heap_allocations** y **stack_allocations**.*

Bloques

*En este código, los bloques están representados por funciones como **addMemoryRecord**, **removeMemoryRecord**, **displayMemoryUsage**, etc. Estas funciones realizan operaciones de gestión de memoria.*

Alcance

*El alcance en este código está relacionado con las variables que tienen un ámbito global (como **heap_allocations**) y local dentro de las funciones. Variables como **heap_allocations** son globales, mientras que los punteros como **current** dentro de la función **displayMemoryUsage** son locales.*

Administración de memoria

*La administración de memoria se maneja mediante las funciones **addMemoryRecord** (para registrar la asignación de memoria dinámica) y **removeMemoryRecord** (para liberar la memoria). El código mantiene un registro de las asignaciones y liberaciones en la memoria heap mediante la estructura **MemoryRecord** y un puntero de lista enlazada (**heap_memory_records**).*

Expresiones

Las expresiones en este código incluyen operaciones como la asignación (=), la comparación (==), y la manipulación de punteros. Un ejemplo es:

```
record->pointer = pointer;
```

Comandos

*Un ejemplo es **malloc()** es un comando para asignar memoria dinámica, y **free()** es un comando para liberar memoria.*

Secuencia

*Aqui un ejemplo también puede ser **malloc()**, la creación del registro en la lista, y la liberación de memoria con **free()** en el orden correcto.*

Selección

*Un ejemplo es el uso de if en el bloque **if (*current)**, donde se verifica si hay un registro en la lista de memoria antes de proceder.*

Iteración

*La iteración se maneja a través de bucles, como el bucle **while (current)** dentro de la función **displayMemoryUsage**, que recorre la lista enlazada de registros de memoria para imprimir las direcciones y tamaños de las asignaciones.*

Subprogramas

*En este código, las funciones **addMemoryRecord removeMemoryRecord**, **displayMemoryUsage**, y otras son ejemplos de subprogramas que realizan tareas específicas relacionadas con la gestión de memoria.*

Tipos de datos

*El código hace uso de tipos de datos como **void*** (punteros), **size_t** (para representar tamaños en bytes), **int** (para contar asignaciones y liberaciones), y **MemoryRecord*** (puntero a estructuras).*

Memory_magment.h

*Y por ultimo analizaremos la libreria de este programa **Memory_magment.h***

Nombres

*El código define varias variables y funciones mediante identificadores, como **heap_allocations**, **heap_deallocations**, **stack_allocations**, **stack_deallocations**, **displayMemoryUsage()***

Marcos de activación

*En este caso, no se muestra directamente un marco de activación típico de una función, pero las funciones como **incrementHeapAllocations()** y **incrementStackAllocations()** se ejecutan cuando se realiza una asignación de memoria, y el marco de activación para cada llamada sería creado en la pila durante su ejecución.*

Alcance

*En este código, los bloques están presentes en la definición de las funciones y las secciones de código condicional. Los contadores de memoria (**heap_allocations**, etc.) tienen un ámbito global, ya que se definen fuera de las funciones.*

Administración de memoria

*ste código maneja la administración de memoria al contar las asignaciones (**heap_allocations**, **stack_allocations**) y las desalocaciones (**heap_deallocations**, **stack_deallocations**) en el montón y la pila. El uso de macros como **MEMORY_MANAGEMENT_DISPLAY** permite habilitar o deshabilitar la visualización de estadísticas de memoria.*

Expresiones

*Las expresiones están presentes en las macros como **displayMemoryUsage()**, que solo tiene efecto si **MEMORY_MANAGEMENT_DISPLAY** está habilitado.*

Comandos

Los comandos en este código son las acciones que toman lugar cuando se llama a las funciones: incrementar contadores de memoria, mostrar estadísticas, etc.

Control de secuencia

El código tiene una estructura condicional en el preprocesador:

```
#if MEMORY_MANAGEMENT_DISPLAY
```

- *Esto permite incluir o excluir ciertas partes del código (como las funciones que manejan la memoria) dependiendo del valor de **MEMORY_MANAGEMENT_DISPLAY**.*

Subprogramas

*Las funciones son definidas como **displayMemoryUsage()**, **incrementHeapAllocations()**, **incrementStackAllocations()**, etc, y se utilizan para realizar tareas específicas, como mostrar el uso de memoria o incrementar los contadores.*

Tipos de datos

*El código utiliza tipos de datos estándar de C, como **int** (para contadores de memoria), **void *** (para punteros de memoria), y **size_t** (para tamaños de bloques de memoria).*

Brandon Aguilar Solis, 14/03/2025