Reporte Practica 3

Autor: Daniel Mojica Salgado

Fecha: 26 de Septiembre de 2025

El objetivo de esta práctica es identificar los elementos fundamentales de los lenguajes de programación: nombres, marcos de activación, bloques de alcance, administración de memoria, expresiones, comandos, control de secuencia como lo es; selección, iteración y recursión, subprogramas, y tipos de datos.

Nombres (identificadores)

Son los nombres que se usan para variables, funciones, estructuras, etc.

```
book_t *library = NULL;
```

- El nombre **library** identifica a la lista de libros.

```
void addBook(book_t **library, int *count);
```

- El nombre **addBook** es un identificador de función.

Marcos de activación (Activation Records / Stack Frames)

Cada vez que se llama a una funcion, se crea un marco de activacion con sus variables locales.

- **En main()**, variables como en el siguiente codigo viven en su marco de activacion

```
main()
{
    int choice = 0;

    //...
}
```

- En issuebook(), los variables locales **bookID, memberID** se crean al entarr la funcion y se destruyen al salir.

```
void issueBook(book_t *library, member_t *members)
{
```

```
    int bookID, memberID;

    //...
}
```

Bloques de alcane (Scopes)

El ambito en el que un nombre es valido

- Dentro de **addBook()**, la variable **book_t new_book* solo existe dentro de esa función.

```
oid addBook(book_t **library, int* count )
{
    // Asignacion de memoria en el heap
    book_t *new_book = (book_t *)malloc(sizeof(book_t));

    //...
}
```

- Dentro del **switch(choice)** en **main()**, cada case tiene su propio bloque de alcance.

```
switch (choice) {
            case 1:
                addBook(&library, &bookCount);
                break;
            case 2:
                displayBooks(library);
                break;
            case 3:
                addMember(&members, &memberCount);
                break;
            case 4:
                issueBook(library, members);
                break;
            case 5:
                returnBook(library, members);
                break;
            case 6:
                displayMembers(members, library);
                break;
            case 7:
                searchMember(members, library);
                break;
            case 8:
                saveLibraryToFile(library, "library.txt");
                saveMembersToFile(members, "members.txt");
                printf("Saliendo del programa\n");
                break;
            default:
```

```
                printf("Esta no es una opcion valida!!!\n");
                break;
        }
```

Administracion de memoria

Se observa un uso de memoria automática, estática y dinámica

- **Atuomatica** (stack):

```
int main() {
    // Variables automaticas (almacenadas en el stack)
    int bookCount = 0, memberCount = 0;
    int choice = 0;
    //...
}
```

- **Dinámica** (heap): **book_t *new_book = (book_t)malloc(sizeof(book_t));* en **addBook()**.

```
void addBook(book_t **library, int* count ) {
    // Asignacion de memoria en el heap
    book_t *new_book = (book_t *)malloc(sizeof(book_t));
    //...
}
```

- **Estática: static int static_var = 0;** declarada al inicio del archivo.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "memory_management.h"

// Variable estatica (almacenada en el segmento de datos)
static int static_var = 0;
```

Expresiones

Combinaciones de variables, valores y operadores.

- **if (current->id == bookID)**: expresión booleana usada en una condición.

```
while (current) {
        if (current->id == bookID) {
            return current;
        }
```

```
        current = current->next;
    }
```

Comandos (sentencias)

Son las intrucciones ejecutables

- **printf();** comando de salida.

```
printf("\nLibro prestado satisfactoriamente!\n");
```

- **return current;** -> comando de retorno de función.

```
void displayBooks(book_t *library) {
    if (!library) {
        printf("\nNo hay libros disponibles.\n"); // comando de salida
        return; // comando de retorno
    }

    printf("\nLibros disponibles en biblioteca:\n");
    displayBooksRecursive(library);
    displayMemoryUsage();
}
```

Control de secuencia

Incluye selección, iteración y recursión.

Selección

- if (bookFound && memberFound) en issueBook().

```
if (bookFound && memberFound) {
        bookFound->quantity--;
        memberFound->issued_count++;
        memberFound->issued_books = realloc(memberFound->issued_books,
memberFound->issued_count * sizeof(int));
        incrementHeapAllocations(memberFound->issued_books, memberFound-
>issued_count * sizeof(int));
        printf("Memoria reasignada para los libros prestados del miembro (ID: %d)
en el heap\n", memberFound->id);
        memberFound->issued_books[memberFound->issued_count - 1] = bookID;
        printf("\nLibro prestado satisfactoriamente!\n");
    } else {
```

```
        printf("\nLibro o miembro no encontrados.\n");
    }
```

Iteración

- **for (int i = 0; i < current->issued_count; i++)** en **displayMembers()**.

```
for (int i = 0; i < current->issued_count; i++) {
            book_t *book = findBookById(library, current->issued_books[i]);
            if (book) {
                printf("  Libro ID: %d\n  Titulo: %s\n  Autor: %s\n", book->id,
book->title, book->author);
            }
}
```

Recursión

- **displayBooksRecursive(library->next);** se llama a sí misma para recorrer los libros.

```
void displayBooksRecursive(book_t *library) {
    if (!library) {
        return;
    }
    printf("\nID libro: %d\nTitulo: %s\nAutor: %s\nAno de publicacion: %d\nGenero:
%s\nCantidad: %d\n",
        library->id, library->title, library->author, library->publication_year,
genreToString(library->genre), library->quantity);
    displayBooksRecursive(library->next);
}
```

Subprogramas (Funciones)

Son las funciones definidas.

```
// Prototipos de funciones
const char* genreToString(genre_t genre);
void addBook(book_t **library, int* count);
book_t* findBookById(book_t *library, int bookID);
```

Tipos de datos

Enumeración

```
typedef enum {
    FICTION,
    NON_FICTION,
    SCIENCE,
    HISTORY,
    FANTASY,
    BIOGRAPHY,
    OTHER
} genre_t;
```

Estructura

```
typedef struct _book {
    int id;
    char title[100];
    char author[100];
    int publication_year;
    genre_t genre;
    int quantity;
    struct _book *next;
} book_t;

typedef struct _member {
    int id;
    char name[100];
    int issued_count;
    int *issued_books;
    struct _member *next;
} member_t;
```

POrtafolio Github: https://github.com/DanyNZ0124/portafolio/tree/master