+++ date = '2025-02-21T10:20:06-08:00' draft = false title = 'Practica1: elementos básicos de los lenguajes de programación' +++

Introducción

En esta practica, analizaremos un programa en C que se nos proporcionó, desglosando los elementos esenciales de los lenguajes de programación. Se identificarán los componentes clave como identificadores, objetos, entornos, bloques, alcance, administración de memoria y estructuras de control. Además, se analizará cómo se maneja la memoria dinámica, estática y automática dentro del programa.

Análisis

1. Nombres (Identificadores)

Los identificadores son nombres usados para variables, funciones y constantes en el código. En este programa, algunos ejemplos son:

- **Variables:** `library`, `members`, `bookCount`, `memberCount`.
- **Funciones:** `addBook`, `displayBooks`, `saveLibraryToFile`, `loadLibraryFromFile`.
- **Constantes:** `FICTION`, `NON_FICTION`, `SCIENCE`, `HISTORY`, `FANTASY`, `BIOGRAPHY`, `OTHER`.

Tambien tenemos otras secciones dentro del programa:

- **MEMORY_MANAGEMENT_H** Es un nombre de archivo de cabecera utilizado para evitar la inclusión múltiple del archivo.
- **MEMORY_MANAGEMENT_DISPLAY** Es una macro de preprocesador utilizada para controlar si se debe mostrar la información sobre el uso de memoria.

- **heap_allocations** Son variables externas que almacenan contadores de las asignaciones y liberaciones de memoria en el heap y la pila.

Estos identificadores hacen que el código sea más comprensible al describir claramente su propósito.

2. Objetos

Un objeto en C es una región de memoria con un tipo de datos. En este programa:

- **Estructuras:** `book_t` y `member_t` almacenan información sobre libros y miembros, respectivamente.
- **Punteros:** Se utilizan para gestionar memoria dinámica, como `book_t *library`.
- **Archivos:** `FILE *` se emplea para manejar la lectura y escritura de datos en archivos.

3. Entornos

El entorno de ejecución del programa contiene variables con distintos ámbitos:

- **Globales:** Variables como `heap_allocations` y `stack_allocations` permiten gestionar la memoria en todo el programa.
- **Locales:** Variables definidas dentro de funciones como `choice` en `main` solo existen dentro de su bloque.
- **Segmento de memoria:** Secciones como BSS almacenan variables no inicializadas.

4. Bloques

Los bloques en C se definen con llaves `{}` y delimitan secciones de código:

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

Estos bloques organizan el programa y definen el ámbito de las variables.

5. Alcance (Scope)

El alcance define la visibilidad de las variables:

- **Global:** Variables accesibles en todo el programa, como `heap_allocations`.
- **Local:** Variables dentro de funciones, como `bookID` en `findBookById`.

6. Administración de Memoria

El programa maneja tres tipos principales de memoria:

Memoria Estática

- Se asigna en tiempo de compilación y persiste durante toda la ejecución del programa.
- **Ejemplo:** `static int static_var = 0;` se almacena en el segmento de datos.

- Se usa en variables globales como `heap_allocations` y `stack_allocations` dentro de `memory_management.c`.

Memoria Automática

- Se asigna en el **stack** y se libera automáticamente al salir del bloque de código donde se declaró.
- **Ejemplo:** En `main()`, variables como `bookCount`, `memberCount` y `choice` son automáticas.

Memoria Dinámica

- Se gestiona manualmente con `malloc`, `realloc` y `free` y se almacena en el **heap**.
- **Funciones que la usan:**
  - `addBook()` asigna memoria a un nuevo libro:

```
book_t *new_book = (book_t *)malloc(sizeof(book_t));
incrementHeapAllocations(new_book, sizeof(book_t));
```

  - `addMember()` asigna memoria a un nuevo miembro.
  - `issueBook()` usa `realloc` para cambiar el tamaño de un arreglo dinámico.
  - `freeLibrary()` y `freeMembers()` liberan memoria previamente asignada.

Ejemplo de liberación de memoria:

```
void freeLibrary(book_t *library) {
    book_t *current = library;
    while (current) {
        book_t *next = current->next;
        incrementHeapDeallocations(current);
        free(current);
        current = next;
    }
    displayMemoryUsage();
}
```

7. Expresiones

Las expresiones incluyen:

- **Comparaciones:** `current->id == bookID`
- **Asignaciones:** `new_book->next = *library`
- **Operaciones matemáticas:** `bookFound->quantity--`

8. Comandos (Sentencias)

Ejemplos de sentencias en el programa:

- Declaración: `int choice = 0;`
- Llamadas a función: `addBook(&library, &bookCount);`
- Retorno de función: `return 0;`

9. Secuencia

El código sigue una ejecución secuencial, salvo por las estructuras de control que alteran el flujo.

10. Selección (Condicionales)

El programa usa `if`, `else` y `switch` para la toma de decisiones. Ejemplo:

```
switch (choice) {
    case 1:
        addBook(&library, &bookCount);
        break;
    case 2:
        displayBooks(library);
        break;
}
```

11. Iteración (Bucles)

Los bucles `while` y `for` permiten repetir código:

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

12. Recursión

Ejemplo de función recursiva para mostrar libros:

```
void displayBooksRecursive(book_t *library) {
    if (!library) return;
    printf("%s\n", library->title);
    displayBooksRecursive(library->next);
}
```

13. Subprogramas (Funciones)

El código se divide en funciones, lo que mejora la organización y reutilización del código.

14. Tipos de Datos

El programa usa:

- **Primitivos:** `int`, `char`, `size_t`.
- **Definidos por el usuario:**
  - `genre_t` - Enumeración de géneros de libros.
  - `book_t` y `member_t` - Estructuras para representar libros y miembros.

Conclusión

El programa en C analizado maneja memoria al combinar el uso de memoria estática, automática y dinámica, lo que optimiza el rendimiento y la organización del código. Usa tipos de datos conocidos e implementa funciones de manera correcta, permitiendome entender cómo se estructura un programa en C.

Referencias

- GeeksforGeeks. (s.f.). Memory Layout of C Programs. Recuperado de: https://www.geeksforgeeks.org/memory-layout-of-c-program/
- TutorialsPoint. (s.f.). C Programming - Data Types. Recuperado de: https://www.tutorialspoint.com/cprogramming/c_data_types.htm
- Delgado, H. (2020). Variables Locales y Globales en C - Diferencias y ejemplo. Recuperado de: https://disenowebakus.net/variables-locales-globales.php