Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Paradigmas de la programación

Práctica 1

Elementos básicos de los lenguajes de programación

Arturo Rafael Cornejo Escobar

1 de abril del 2025

INTRODUCCIÓN

En el ámbito de la programación, los **nombres** son elementos fundamentales para la abstracción. Estos permiten hacer referencia a cualquier entidad en diversos contextos del código y una entidad puede contar con varios nombres que la referencian.

Cuando se hace mención del contexto del código, es debido que a que una característica esencial de un nombre es que debe ser limitado a una región de **alcance**.

El alcance es una característica que define el contexto del que un nombre referencia, es decir, el lugar donde se buscara ese nombre.

Los **bloques** son partes del código que organizan nombres con un alcance local y estos corresponden a un marco en el entorno. En la programación, es usual la existencia de bloques de función y los bloques anidados donde los nombres dentro de estos bloques solo son visibles dentro de ellos.

Un nombre está **sobrecargado** si refiere a múltiples entidades en el mismo alcance. Algunos lenguajes resuelven esto basándose en cómo se usa el nombre.

Los **marcos de activación** son espacios de memoria temporales que se crean cada vez que se llama a una función.

Contienen la información esencial para que esa función se ejecute correctamente, como:

- **Parámetros:** Los valores que se le pasaron.
- **Variables locales:** Las variables que la función declara para su uso interno.
- **Dirección de retorno:** Dónde debe continuar el programa cuando la función termina.

Se apilan uno encima del otro en la **pila de llamadas** a medida que se invocan funciones, y se eliminan cuando cada función finaliza, liberando esa memoria. Esto permite que las funciones se ejecuten de forma independiente

El objetivo de esta práctica es identificar los **elementos esenciales de los lenguajes de programación**, incluyendo: **nombres**, **marcos de activación**, **bloques de alcance**, **administración de memoria**,

expresiones, **comandos**, **control de secuencia** (selección, iteración y recursión), **subprogramas** y **tipos de datos**.

DESARROLLO

Nombres

Variables: `static_var`, `bss_var`, `bookCount`, `memberCount`, `choice`, `bookID`, `memberID`, `genre`, `found`, `i`, `j`.

Tipos de variable personalizada: `genre_t`, `book_t`, `member_t`. Etos son nombres utilizados para definir datos abstractos como lo serían las estructuras y las enumeraciones. En este caso, la primera es el nombre para un tipo de dato de enumeración y el resto son estructuras.

Miembros de tipo de dato abstracto: `id`, `title`, `author`, `publication_year`, `genre`, `quantity`, `next` (para `book_t`) y `id`, `name`, `issued_count`, `issued_books`, `next` (para `member_t`). Estos **nombres** corresponden a campos individuales de datos de la estructura, no son variables y son usados para acceder a esos campos.

Funciones: `genreToString`, `addBook`, `findBookById`, `displayBooksRecursive`, `displayBooks`, `addMember`, `issueBook`, `returnBook`, `freeLibrary`, `freeMembers`, `saveLibraryToFile`, `loadLibraryFromFile`, `saveMembersToFile`, `loadMembersFromFile`, `displayMembers`, `searchMember`, `main`. Cada una de estas funciones tiene un **nombre** que te permite llamarlas.

Punteros: `library`, `members`, `new_book`, `new_member`, `current`, `bookFound`, `memberFound`, `current_book`, `current_member`, `file`, `book`. Estos son **nombres** que "apuntan" a ubicaciones en la memoria donde se almacenan datos.

Constantes (Enumeradores): `FICTION`, `NON_FICTION`, `SCIENCE`, `HISTORY`, `FANTASY`, `BIOGRAPHY`, `OTHER`. Son **nombres** para representar valores constantes y que sea más eficiente la referencia.

Bloques de Alcance

Alcance globlal:

- **Variables:** `static_var`, `bss_var`, `genre_t`, `book_t`, `member_t`, y los **prototipos de las funciones**.
- **Visibilidad:** Estos nombres son accesibles desde cualquier parte del código, dentro de cualquier función.
- `static_var` tiene la palabra clave `static`, que restringe su visibilidad solo a este archivo, por lo que no puede ser accedida desde otros archivos.

Bloques de función: Cada función, como `main()`, `addBook()`, o `issueBook()`, establece su propio **bloque principal**, delimitado por llaves (`{}`). Dentro de este bloque, se definen **nombres** locales que son específicos de esa función:

- **Parámetros de la función:** Son nombres que representan los datos que se pasan a la función cuando esta es invocada. En la función `addBook(book_t **library, int* count)`, `library` y `count` son nombres que existen únicamente dentro del ámbito local de la función misma.

- **Variables locales:** Son nombres que se declaran y utilizan internamente dentro del cuerpo de una función. Por ejemplo, si `new_book` se declara dentro de `addBook()`, o si `library`, `members`, `bookCount`, `memberCount`, y `choice` se declaran dentro de `main()`, estos nombres son únicos y están limitados al alcance dentro la función.

Bloques anidados:

Dentro de las funciones, las estructuras como `if`, `else`, `while`, `for`, y `switch` también pueden tener sus propios bloques, con llaves `{}`. **Ejemplo en `returnBook()`:**

```
if (bookFound && memberFound) { // <-- Inicio de un bloque anidado
    int found =  0; // 'found' solo existe dentro de este bloque
    for (int i =  0; i <  memberFound->issued_count; i++) { // <-- Inicio de otro
bloque anidado
        // 'i' solo existe aquí
        for (int j = i; j <  memberFound->issued_count  -  1; j++) {
            // 'j' solo existe en este bloque más interno mientras que como i es
de un bloque más externo también existe.
        }
    }
}
```

Marcos de activación:

`main()` y su Marco: Cuando `main()` comienza, se crea un marco de activación. Dentro de este marco, se guardan los nombres `library`, `members`, `bookCount`, `memberCount`, `choice` y sus valores.

Llamadas a `addBook()` o `issueBook()`: Cada vez que llamas a una de estas funciones, se crea un **nuevo marco de activación** encima del marco de `main()`. Este nuevo marco guarda:

- Los **parámetros** que se le pasaron a la función (ej. `library` y `count` en `addBook`).
- Las **variables locales** declaradas dentro de esa función (ej. `new_book` en `addBook`, `bookID` y `memberID` en `issueBook`).

Tipos de Datos

El código utiliza y define varios tipos de datos para representar la información de la biblioteca.

Tipos:

- **int**: Utilizado para identificadores, años de publicación, cantidades, contadores (bookCount, memberCount, issued_count), opciones de menú y variables de bucle (i, j).
- **char**: Utilizado en arreglos para almacenar cadenas de texto como títulos, autores, nombres y géneros (title[100], author[100], name[100], genre_str[50]).
- **FILE***: Puntero a un tipo de dato que representa un flujo de archivo, utilizado para operaciones de entrada/salida.
- **void**: Se usa como tipo de retorno para funciones que no devuelven ningún valor y para punteros genéricos (void*) antes de la conversión explícita ((book_t *), (member_t *), (int *)).

Tipos Personalizados:

- **Enumeración (enum)**: `genre_t` define un conjunto de constantes enteras con nombres significativos (FICTION, NON_FICTION, SCIENCE, HISTORY, FANTASY, BIOGRAPHY, OTHER) para representar los géneros de los libros.
- **Estructuras (struct)**:
  - `book_t`: Agrupa datos relacionados con un libro (id, title, author, publication_year, genre, quantity) y un puntero a la siguiente estructura en una lista enlazada (*next).
  - `member_t`: Agrupa datos relacionados con un miembro (id, name, issued_count, *issued_books) y un puntero a la siguiente estructura en una lista enlazada (*next).

Administración de Memoria El programa demuestra un control explícito sobre la administración de memoria, utilizando diferentes segmentos y operaciones de asignación y liberación.

Segmento de Datos:

- `static int static_var = 0;`: Una variable estática inicializada, almacenada en el segmento de datos. Su vida útil es la duración del programa.

Segmento BSS:

- `int bss_var;`: Una variable global no inicializada, almacenada en el segmento BSS (Block Started by Symbol). Su vida útil es la duración del programa.

Pila (Stack):

- Variables locales dentro de las funciones (library, members, bookCount, memberCount, choice en main(), bookID, memberID en issueBook(), new_book como puntero, genre en addBook(), found, i, j en returnBook()). Estas variables se asignan y liberan automáticamente cuando la función entra y sale de su ámbito.

Montón (Heap):

- **Asignación Dinámica**: Se utiliza `malloc()` para asignar memoria para nuevas estructuras `book_t` y `member_t` (`new_book = (book_t *)malloc(sizeof(book_t))`, `new_member = (member_t *)malloc(sizeof(member_t))`, `new_member->issued_books = (int *)malloc(new_member->issued_count * sizeof(int))`).
- **Reasignación Dinámica**: Se emplea `realloc()` para ajustar el tamaño del arreglo `issued_books` cuando es decrementado un libro al ser prestado a un miembro (`memberFound->issued_books = realloc(...)`).
- **Liberación Dinámica**: Se usa `free()` para liberar explícitamente la memoria asignada en el heap, previniendo fugas de memoria (`free(current)`, `free(current->issued_books)`). Las funciones `freeLibrary()` y `freeMembers()` son responsables de esta tarea al finalizar el programa.

Expresiones Las expresiones son combinaciones de operadores y operandos que producen un valor.

Expresiones Aritméticas:

- `sizeof(book_t)`: Calcula el tamaño en bytes de la estructura `book_t`.

- `memberFound->issued_count * sizeof(int)`: Calcula el tamaño requerido para un arreglo de enteros.
- `(*count)++`: Incrementa el valor apuntado por `count`.
- `bookFound->quantity--`: Decrementa la cantidad que representa un libro.

Expresiones Relacionales:

- `!new_book`: Verifica si un puntero es nulo.
- `current->id == bookID`: Compara valores que representan identificadores.
- `current_book->quantity > 0`: Verifica si el valor es mayor a 0, la abstracción sería que si hay libros.
- `strcmp(genre_str, "Ficcion") == 0`: Compara cadenas de texto.
- `choice != 8`: Condición de salida del bucle `do-while`.

Expresiones Lógicas:

- `bookFound && memberFound`: Comprueba si ambos punteros son válidos (no nulos).

Expresiones de Asignación:

- `new_book->next = *library;`: Asigna la dirección del puntero a puntero desferenciado `*library` al puntero `next` de `new_book`.
- `*library = new_book;`: Actualiza el puntero `library` para que apunte al nuevo libro.
- `bookFound->quantity = 4;`: Asigna un nuevo valor.

Expresiones de Conversión de Tipo:

- `(book_t *)malloc()`: Convierte el puntero `void*` devuelto por `malloc` a un puntero de tipo `book_t*`.
- `(genre_t)genre`: Convierte un entero a un tipo de enumeración `genre_t`.

Expresiones de Acceso a Miembros:

- `current->id`: Accede al miembro `id` de la estructura apuntada por `current`.
- `memberFound->issued_books[i]`: Accede a un elemento específico del arreglo `issued_books` dentro de la estructura.

Comandos (Sentencias) Los comandos (o sentencias) son instrucciones que realizan una acción.

- **Declaración de Variables**: `int bookID, memberID;`
- **Asignación**: `static_var = 0;`, `new_book->id = 1;`
- **Llamadas a Función**: `printf()`, `scanf()`, `malloc()`, `free()`, `addBook()`, `displayMemoryUsage()`, `fopen()`, `fclose()`.
- **Sentencias de Retorno**: `return;` (en funciones void), `return NULL;` (en findBookById), `return 0;` (en main).

Control de Secuencia El control de secuencia determina el orden en que se ejecutan los comandos.

Selección (Decisión)

- **if-else**:
  - `if (!new_book) { ... return; }`: Manejo de errores de asignación de memoria.

    - `if (current->id == bookID && current->quantity > 0) { ... break; }`: Condición para encontrar un libro disponible.
    - `if (bookFound && memberFound) { ... } else { ... }`: Rama principal de decisión para prestar/devolver libros.
    - `if (found) { ... } else { ... }`: Determina si el libro fue encontrado para devolver.
    - `if (!library) { ... return; }`: Verifica si la lista de libros está vacía.
- **switch-case**:
    - En `genreToString()`: Selecciona la cadena de texto de un género según el valor de la enumeración.
    - En `main()`: Controla el flujo del menú principal basándose en la opción (`choice`) seleccionada por el usuario. Cada `case` representa una opción de menú, con un `default` para opciones inválidas.

Iteración (Bucles)

- **while**:
    - `while (current)`: Utilizado para recorrer listas enlazadas de libros y miembros (`findBookById`, `displayMembers`, `freeLibrary`, `freeMembers`, `saveLibraryToFile`, `loadLibraryFromFile`, `saveMembersToFile`, `loadMembersFromFile`, `searchMember`).
    - `while (!feof(file))`: En las funciones de carga de archivos, este bucle lee el archivo hasta el final.
- **for**:
    - `for (int i = 0; i < memberFound->issued_count; i++)`: Itera sobre el arreglo de libros prestados por un miembro (`returnBook`, `saveMembersToFile`, `loadMembersFromFile`, `displayMembers`, `searchMember`).
    - `for (int j = i; j < memberFound->issued_count - 1; j++)`: Bucle anidado para desplazar elementos en el arreglo al devolver un libro.
- **do-while**:
    - `do { ... } while(choice != 8);`: En `main()`, este bucle garantiza que el menú se muestre al menos una vez y se repita hasta que el usuario elija la opción de salir (8).

Recursión

- `displayBooksRecursive(book_t *library)`: Esta función demuestra recursión.
    - **Caso Base**: `if (!library) { return; }`: La recursión termina cuando el puntero `library` es nulo (es decir, se ha llegado al final de la lista enlazada de libros).
    - **Paso Recursivo**: `displayBooksRecursive(library->next);`: La función se llama a sí misma con el siguiente nodo en la lista, procesando cada libro en orden. Esto permite imprimir la información de todos los libros en la lista.

Referencias

Cooper, K. D., & Torczon, L. (2012). **The procedure abstraction**. En *Elsevier eBooks* (pp. 269-330). https://doi.org/10.1016/b978-0-12-088478-0.00006-2

Alcance. (s. f.). JSvis. https://centrogeo.github.io/JSvis/13-Alcance.html

TylerMSFT. (s. f.). **Nombres representativos**. Microsoft Learn. https://learn.microsoft.com/es-es/cpp/build/reference/decorated-names?view=msvc-170

IBM Debug for z/OS. (s. f.). **Expresiones en C**. IBM. https://www.ibm.com/docs/es/debug-for-zos/15.0.x?topic=programs-c-c-expressions

---

Dirección del repositorio:

https://github.com/roixarturo/portafolio1

Dirección de la página de GitHub:

https://roixarturo.github.io/portafolio1/