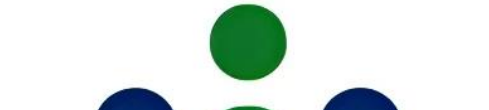


Ejercicio 24

Docente: Jimmy Nataniel Requena Llorentty

Materia: Programación III

Carrera: Ingeniería En Sistemas

Estudiantes: Joaquin Marcos Maita Flores

Santa Cruz – Bolivia

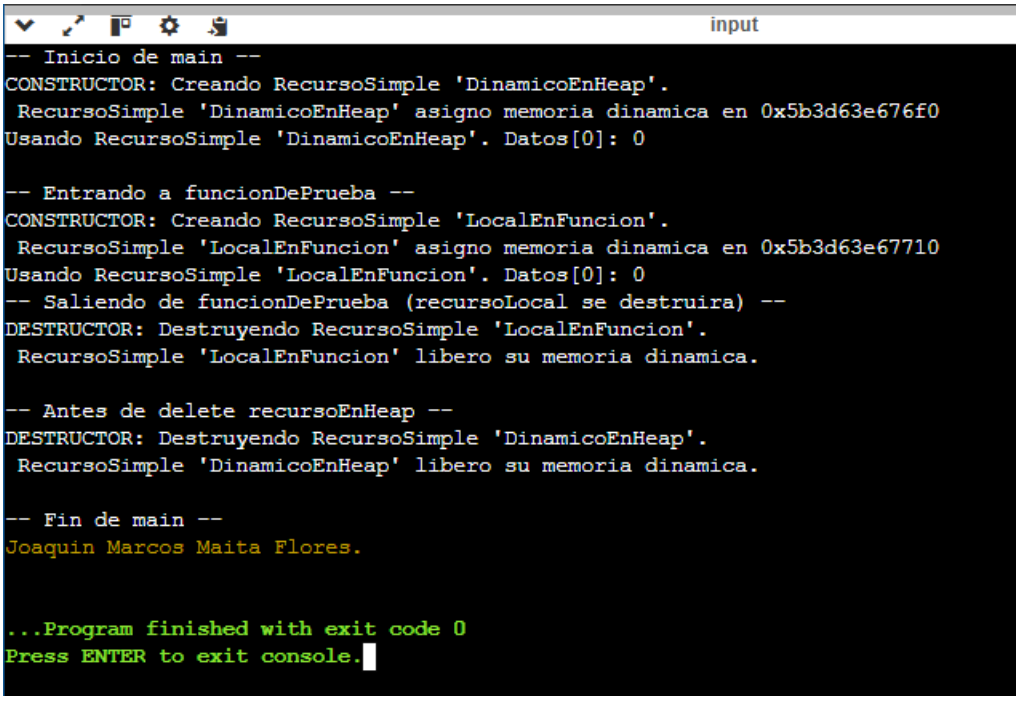
2025

CICLO DE VIDA EN ACCIÓN: RECURSOSIMPLE

```cpp
#include <iostream>
#include <string>

class RecursoSimple {
private:
    std::string nombreRecurso;  // [1] Variable miembro para almacenar el nombre del recurso
    int* datosDinamicos;       // [2] Puntero para memoria dinámica (gestión manual)

public:
    // [3] Constructor - Se ejecuta al crear un objeto
    RecursoSimple(const std::string& nombre) : nombreRecurso(nombre) {
        std::cout << "CONSTRUCTOR: Creando RecursoSimple '" << nombreRecurso << "'." << std::endl;
        // [4] Asignación de memoria dinámica (5 enteros)
        datosDinamicos = new int[5];
        std::cout << " RecursoSimple '" << nombreRecurso << "' asigno memoria dinamica en "
                  << datosDinamicos << std::endl;
        // [5] Inicialización de los valores
        for (int i = 0; i < 5; ++i)
            datosDinamicos[i] = i * 10;
    }

    // [6] Destructor - Se ejecuta al destruir el objeto
    ~RecursoSimple() {
        std::cout << "DESTRUCTOR: Destruyendo RecursoSimple '" << nombreRecurso << "'." << std::endl;
        // [7] Liberación de memoria dinámica (CRUCIAL para evitar leaks)
        delete[] datosDinamicos;
        datosDinamicos = nullptr;  // [8] Buena práctica (evita dangling pointers)
        std::cout << " RecursoSimple '" << nombreRecurso << "' libero su memoria dinamica." << std::endl;
    }

    // [9] Método para usar el recurso
    void usarRecurso() const {
        std::cout << "Usando RecursoSimple '" << nombreRecurso << "'. Datos[0]: "
                  << (datosDinamicos ? datosDinamicos[0] : -1) << std::endl;
    }
};
```

```cpp
// [10] Función que demuestra el ciclo de vida de un objeto local
void funcionDePrueba() {
    std::cout << "\n-- Entrando a funcionDePrueba --" << std::endl;
    RecursoSimple recursoLocal("LocalEnFuncion");  // [11] Objeto en stack
(destrucción automática)
    recursoLocal.usarRecurso();
    std::cout << "-- Saliendo de funcionDePrueba (recursoLocal se destruira) --"
<< std::endl;
}

int main() {
    std::cout << "-- Inicio de main --" << std::endl;
    RecursoSimple* recursoEnHeap = nullptr;  // [12] Puntero para objeto en heap

    // [13] Creación de objeto en heap (gestión manual)
    recursoEnHeap = new RecursoSimple("DinamicoEnHeap");
    if (recursoEnHeap) {
        recursoEnHeap->usarRecurso();
    }

    funcionDePrueba();  // [14] Llamada a función con objeto local

    // [15] Liberación explícita de memoria en heap
    std::cout << "\n-- Antes de delete recursoEnHeap --" << std::endl;
    delete recursoEnHeap;  // [16] Llama al destructor
    recursoEnHeap = nullptr;

    std::cout << "\n-- Fin de main --" << std::endl;
    std::cout << "\033[33mJoaquin Marcos Maita Flores.\033[0m" << std::endl;
    return 0;
}
```

CÓDIGO EJECUTADO



Este ejemplo enseña:

1. **Gestión de memoria:** Demuestra asignación (new) y liberación (delete) manual de memoria.
2. **Ciclo de vida**: Muestra cuándo se llaman constructores/destructores (objetos en stack vs heap).
3. **RAII**: El destructor libera recursos automáticamente (evita leaks).
4. Seguridad: Uso de nullptr y verificación de punteros.

Anexo

https://onlinegdb.com/S9a2wb7zi