Informe de Implementación: Manejo de Memoria en C++

Curso: Sistemas Operativos
Autor: Jean Karlo Buitrago Orozco
Fecha: Octubre 2025

Pontificia Universidad Javeriana Cali

Introducción

El presente informe describe el desarrollo e implementación de una serie de ejercicios enfocados en el manejo de memoria en el lenguaje C++. Estas actividades fueron tomadas del material guía *Classwork 3* y tienen como objetivo comprender cómo funcionan los punteros, las referencias, los arreglos, la memoria dinámica y las diferentes áreas de almacenamiento (*stack*, *heap* y *code segment*).

Cada sección presenta la actividad implementada, su funcionamiento y los resultados observados al ejecutar el programa `actividades.cpp`.

Actividad 1: Variables y Punteros

Esta actividad introduce los conceptos básicos de dirección de memoria y acceso indirecto mediante punteros.

Descripción: Se declara una variable entera `numero` y se muestra su dirección de memoria. Luego, se crea un puntero que apunta a esta variable, modificando su valor indirectamente mediante `*ptr = 100;`.

Salida observada:

```
========== ACTIVIDAD 1 ==========
Variables y Punteros

Variable numero = 42
Direccion de memoria de numero: 0x61fed8

Despues de modificar con puntero:
Nuevo valor de numero = 100
Direccion de memoria de numero: 0x61fed8
Direccion almacenada en ptr: 0x61fed8
```

El resultado demuestra cómo un puntero puede acceder y modificar directamente el valor de una variable en memoria.

Actividad 2: Punteros y Referencias

Esta actividad compara el uso de punteros con el de referencias para modificar el valor de una variable.

Explicación: El puntero requiere desreferenciarse con * para acceder al valor, mientras que la referencia actúa como un alias directo. Ambas formas modifican la misma posición de memoria.

Salida observada:

```
========== ACTIVIDAD 2 ==========
Punteros y Referencias

Valor inicial: 25
Direccion de valor: 0x61fed4

Despues de modificar con puntero:
Nuevo valor: 50

Despues de modificar con referencia:
Valor final: 75

Direcciones de memoria:
Direccion de valor: 0x61fed4
Direccion almacenada en puntero: 0x61fed4
Direccion de referencia: 0x61fed4
```

Actividad 3: Arrays y Punteros

Aquí se demuestra que un arreglo y un puntero comparten una relación directa en C++.
Salida observada:

```
========== ACTIVIDAD 3 ==========
Arrays y Punteros

Array inicial: 10 20 30 40 50
Array modificado con punteros: 100 20 300 40 500

Direcciones de memoria:
Direccion del array (primer elemento): 0x61febc
Direccion del puntero: 0x61febc

Direcciones de cada elemento:
numeros[0] en 0x61febc = 100
numeros[1] en 0x61fec0 = 20
numeros[2] en 0x61fec4 = 300
numeros[3] en 0x61fec8 = 40
numeros[4] en 0x61fecc = 500
```

Análisis: Cada posición del arreglo puede ser accedida sumando un desplazamiento al puntero base. El ejercicio muestra cómo manipular elementos de un arreglo usando aritmética de punteros.

Actividad 4: Memoria Dinámica – Matriz 2D

El objetivo es practicar la reserva y liberación de memoria dinámica con el operador `new` y `delete`.

Salida observada:

```
========== ACTIVIDAD 4 ==========
Asignacion Dinamica - Matriz 2D

Llenando la matriz con valores...

Matriz 3x4:
1 2 3 4
5 6 7 8
9 10 11 12

Direcciones de memoria de las filas:
Fila 0: 0x1097fd0
Fila 1: 0x1097f08
Fila 2: 0x1097f20

Memoria liberada correctamente.
```

Análisis: Cada fila de la matriz se reserva dinámicamente, mostrando cómo se crean estructuras de datos flexibles en tiempo de ejecución. La liberación manual es crucial para evitar fugas de memoria.

Actividad Extra: Stack, Heap y Code

Se estudian las tres principales regiones de memoria en tiempo de ejecución: **Stack**, **Heap** y **Code**.

Salida observada:

```
========== ACTIVIDAD EXTRA ==========
Stack, Heap y Code en C++

=== STACK (Variables locales) ===
variableLocal en: 0x61fed4
otraVariable en: 0x61fed0

=== HEAP (Memoria dinamica) ===
ptrHeap1 apunta a: 0x1097fb8 (valor: 100)
ptrHeap2 apunta a: 0x1097fc8 (valor: 200)

=== CODE (Segmento de codigo) ===
Direccion de funcionEjemplo(): 0x401db4
Direccion de actividad1(): 0x401460
Direccion de actividad2(): 0x4015df

=== ANALISIS ===
Las variables del stack suelen estar en direcciones altas
El heap esta en direcciones intermedias
El codigo esta en direcciones bajas
```

Conclusión parcial: Las direcciones del *stack* suelen ser altas, las del *heap* intermedias y las del *segmento de código* bajas, lo que permite diferenciar sus ubicaciones en memoria.

Resultados y Análisis General

A través de las cinco actividades, se pudo observar cómo los punteros y referencias permiten manipular directamente los datos en memoria, comprender las diferencias entre asignación estática y dinámica, y analizar el comportamiento de las direcciones en las diferentes áreas del programa.

El uso de `new` y `delete` demostró la necesidad de una correcta gestión de memoria para evitar fugas. Asimismo, la observación del *stack* y *heap* permitió entender cómo el sistema organiza las variables y funciones durante la ejecución.

Conclusiones

- Se comprendió el funcionamiento de punteros y referencias como herramientas esenciales del manejo de memoria en C++.
- Se aplicó la asignación dinámica de memoria para construir estructuras bidimensionales.

- Se identificaron las regiones principales de memoria y su comportamiento relativo en la ejecución.
- El uso correcto de `delete` y la comprensión de la aritmética de punteros son fundamentales para programas eficientes y seguros.

Repositorio del Proyecto

El código fuente completo del programa se encuentra disponible en el siguiente repositorio de GitHub:

https://github.com/JeanK4/classwork2-operating-systems.git