Created by Dmytro Krainyk

[All code is in my repo on github]

Assignment I: Getting things in order

Task 2 Report: “Dangerous Minds” (Custom Vector)

Introduction

In this task, I implemented a custom dynamic array container (`MyVector`) from scratch to understand the low-level mechanics of memory management in C++. The goal was to replicate the behavior of `std::vector`, specifically its ability to resize dynamically while maintaining contiguous memory layout.

I compared my implementation against:

- `std::vector` (The gold standard).
- `std::list` (To demonstrate the cost of non-contiguous memory).

Implementation Details

1. Manual Memory Management

Unlike high-level containers that manage memory automatically, I used raw pointers (`int* data`) and explicit allocation (`new`, `delete[]`). This gave me granular control over exactly when and how memory is allocated.

2. Table-Doubling Strategy

The critical part of a dynamic array is handling the `push_back` operation when the buffer is full. I implemented the Table Doubling strategy:

- **Mechanism:** When `size == capacity`, I allocate a new block of memory that is twice the size of the current one.
- **Why it matters:** If we only increased capacity by 1, we would have to copy all elements every time, leading to $O(N^2)$ complexity. Doubling ensures that we reuse the existing capacity for as long as possible, resulting in amortized $O(1)$ complexity for insertions.

Code Snippet: Geometric Resizing

```
void push_back(int value) {
    if (length == capacity) {
        // Geometric expansion: Capacity x2
        size_t newCapacity = (capacity == 0) ? 1 : capacity * 2;
        int* newData = new int[newCapacity];

        // Copy old elements to new buffer
        for (size_t i = 0; i < length; i++) newData[i] = data[i];

        delete[] data; // Free old memory
```

```
            data = newData;
            capacity = newCapacity;
        }
        data[length++] = value;
    }
```