# Отчёт

Отчет по программной работе №1 и №2

Тема: Использование динамических массивов и умных указателей в C++

Часть 1: Работа с динамическими массивами (std::vector)

В первой части задания был создан проект "MemoryTrick", в котором исследовалось поведение динамического массива std::vector. Используя метод push\_back, было выполнено 32 добавления элемента в вектор, что позволило наблюдать изменения в размере и вместимости массива.

Правило изменения вместимости: Вместимость вектора увеличивается по экспоненциальному правилу, обычно удваивая текущую вместимость, когда достигается предел. Это поведение предотвращает частые реаллокации и копирование данных, что значительно повышает производительность при добавлении большого количества элементов.

Анализ эффективности: Изменение вместимости std::vector обычно происходит по экспоненциальному правилу, где вместимость удваивается при необходимости расширения. Это правило помогает минимизировать количество реаллокаций, которые могли бы произойти, если бы вместимость увеличивалась наивным способом — по одному элементу за раз.

График изменения вместимости показывает, что с каждым увеличением размера вектора, вместимость увеличивается более чем пропорционально.

Часть 2: Управление памятью с помощью умных указателей (std::shared\_ptr)

Во второй части был создан проект "SmartPtrs", где демонстрировалось использование умных указателей для управления памятью. Были созданы функции foo() и goo(), возвращающие shared\_ptr и обычный указатель соответственно.

Наблюдения: Использование shared\_ptr автоматически управляет памятью, уничтожая объект, когда последний shared\_ptr к нему уничтожается. Это избавляет от необходимости явного вызова delete, предотвращая утечки памяти и ошибки доступа после освобождения.

Сравнение с goo(): При использовании обычного указателя, возвращенного функцией goo(), ответственность за уничтожение объекта ложится на разработчика, что увеличивает риск утечек памяти.

Заключение

Использование умных указателей (std::shared\_ptr) позволяет безопасно и эффективно управлять динамической памятью, автоматизируя процесс очистки и предотвращая многие распространенные ошибки управления ресурсами в C++. Эта техника особенно полезна в больших и сложных проектах, где риск ошибок в управлении памятью особенно высок.