**Лаба5.1**

В ходе выполнения задания по созданию классического приложения Windows была поставлена задача переписать функцию WndProc, избавившись от привычного оператора switch, который традиционно используется для обработки различных сообщений окна. Вместо этого требовалось применить объектно-ориентированный подход, используя полиморфизм и классы, а также максимально ограничить использование оператора if (разрешив лишь один единоразовый случай).

Данная задача представляет интерес, поскольку она демонстрирует, как можно адаптировать классический процедурный стиль программирования в Windows API к современным принципам ООП. В реализации была создана иерархия классов-обработчиков сообщений, каждый из которых отвечает за конкретный тип события (например, создание окна, отрисовка, закрытие и пр.). Вместо большого switch-блока теперь имеется контейнер с объектами-обработчиками, которые вызываются последовательно или выбираются по типу сообщения.

Для выбора нужного обработчика в функции WndProc использовался единственный оператор if для проверки типа сообщения и вызова соответствующего объекта. Такой подход улучшает читаемость и расширяемость кода, позволяя легко добавлять новые обработчики сообщений без изменения централизованной функции. Кроме того, классический switch заменён на механизм виртуальных функций, что соответствует принципам инкапсуляции и полиморфизма.

Оставление switch внутри обработки WM\_COMMAND оправдано тем, что данный блок часто содержит большое количество пунктов меню, и переписывать его в объектно-ориентированном стиле требует значительных усилий и не всегда оправдано с точки зрения эффективности.

**Лаба5.2**

В представленном коде реализованы два класса — базовый A и производный B, оба имеющие конструкторы и деструкторы с выводом отладочных сообщений. Классы управляют строковым полем s в классе A. Создаются объекты динамически и помещаются в список указателей list<A\*> l.  
В функции main объекты создаются и добавляются в список, однако затем отсутствует корректное копирование этих указателей в вектор vector<A\*> v. Также вызывается функция del, которая должна освободить память, но она вызывается для пустого вектора — так как копирование не реализовано, вектор пуст, и удаление не происходит. Это приводит к утечкам памяти, поскольку объекты в списке остаются не удалёнными.  
Ошибки и недочёты:  
1 Отсутствует копирование из списка l в вектор v — нужно перебрать элементы списка и добавить их в вектор.  
 2 Функция del принимает вектор по значению — это создаёт копию вектора, что неэффективно и не влияет на исходный контейнер.  
 3 Отсутствует освобождение памяти объектов из списка l — вызывается удаление только для вектора.  
 4 Неочевидность владения памятью — в программе и список, и вектор владеют указателями, что опасно без явного управления.