**Bubble sorting**

#include <iostream>

#include <vector>

#include <algorithm>

#include <chrono>

#include <cstdlib>

int main() {

const int size = 100000;

std::vector<int> numbers(size);

// Initialize the vector with random values in descending order

for (int i = 0; i < size; ++i) {

numbers[i] = size - i;

}

// Measure the execution time of Bubble Sort

auto startTimeBubbleSort = std::chrono::high\_resolution\_clock::now();

std::sort(numbers.begin(), numbers.end());

auto endTimeBubbleSort = std::chrono::high\_resolution\_clock::now();

auto durationBubbleSort = std::chrono::duration\_cast<std::chrono::microseconds>(endTimeBubbleSort - startTimeBubbleSort);

// Print execution time for Bubble Sort

std::cout << "Bubble Sort Execution Time: " << durationBubbleSort.count() << " microseconds\n";

// Print first 10 and last 10 integers

std::cout << "First 10 integers: ";

for (int i = 0; i < 10; ++i) {

std::cout << numbers[i] << " ";

}

std::cout << "\nLast 10 integers: ";

for (int i = size - 10; i < size; ++i) {

std::cout << numbers[i] << " ";

}

std::cout << "\n";

// Measure the execution time of STL sort

auto startTimeSTLSort = std::chrono::high\_resolution\_clock::now();

std::sort(numbers.begin(), numbers.end());

auto endTimeSTLSort = std::chrono::high\_resolution\_clock::now();

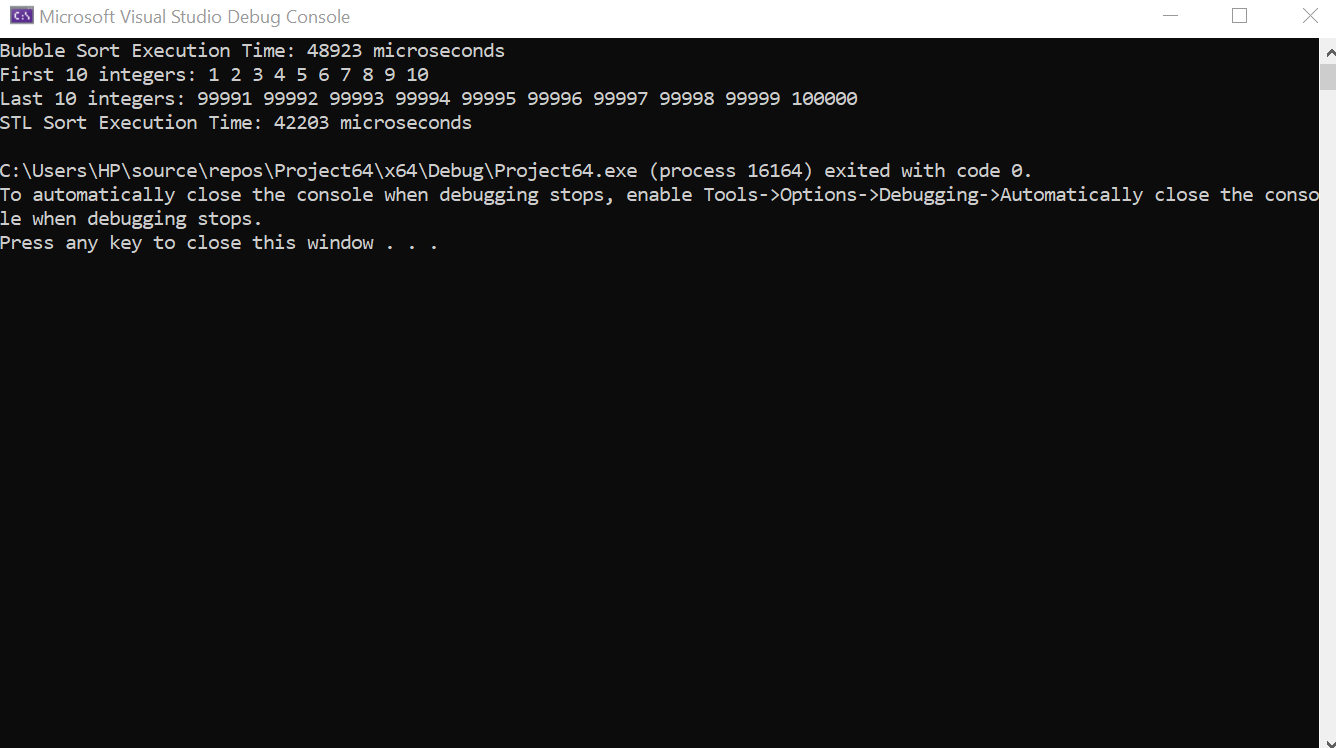
auto durationSTLSort = std::chrono::duration\_cast<std::chrono::microseconds>(endTimeSTLSort - startTimeSTLSort);

// Print execution time for STL sort

std::cout << "STL Sort Execution Time: " << durationSTLSort.count() << " microseconds\n";

return 0;

}



**Inventory Mangement:**

#include <iostream>

#include <vector>

struct Product {

int id;

std::string name;

double price;

};

class InventoryManager {

private:

std::vector<Product> inventory;

public:

void addProduct(const Product& newProduct) {

inventory.push\_back(newProduct);

}

void removeProduct(int productId) {

auto it = std::remove\_if(inventory.begin(), inventory.end(),

[productId](const Product& p) { return p.id == productId; });

inventory.erase(it, inventory.end());

}

void displayInventory() const {

std::cout << "Current Inventory:\n";

for (const auto& product : inventory) {

std::cout << "ID: " << product.id << ", Name: " << product.name << ", Price: " << product.price << "\n";

}

std::cout << std::endl;

}

};

int main() {

InventoryManager inventoryManager;

Product product1 = { 1, "Laptop", 1200.50 };

Product product2 = { 2, "Smartphone", 699.99 };

inventoryManager.addProduct(product1);

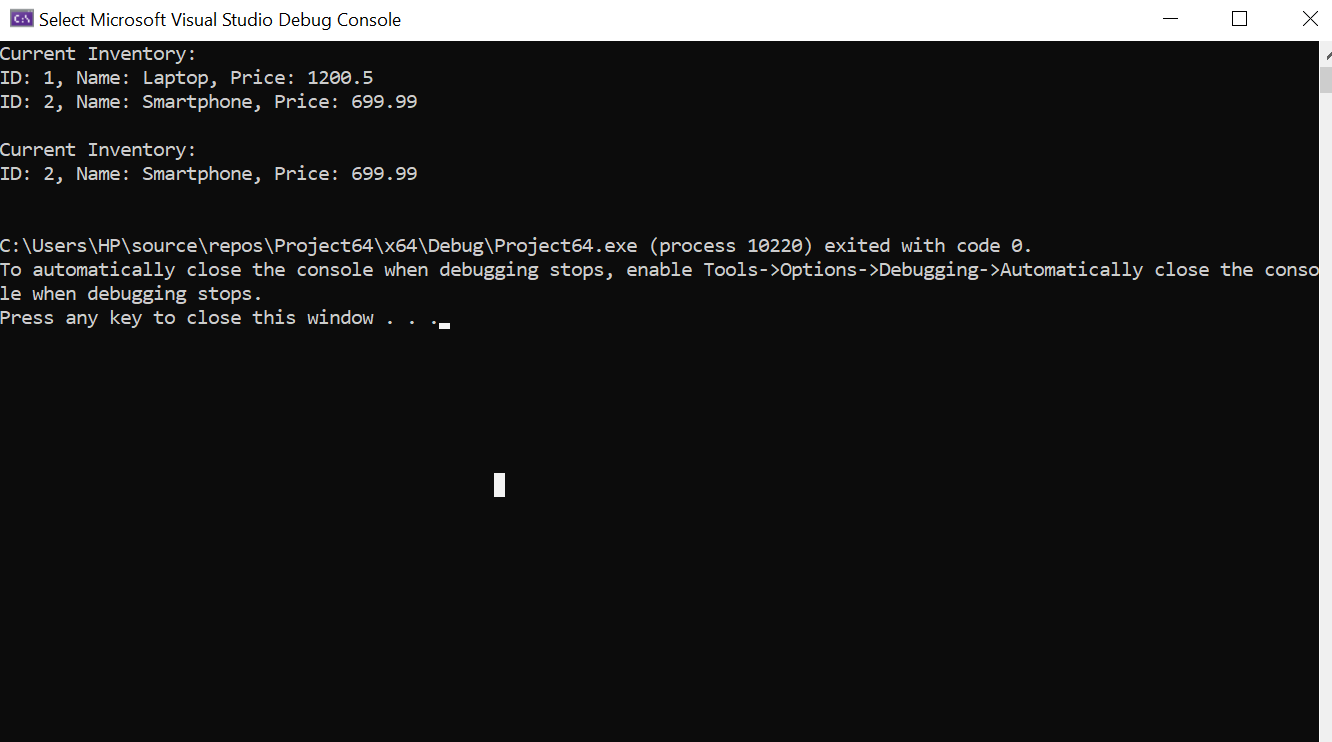
inventoryManager.addProduct(product2);

inventoryManager.displayInventory();

inventoryManager.removeProduct(1);

inventoryManager.displayInventory();

return 0;

****