Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Об’єктно-орієнтоване програмування

Лабораторна робота №8
Тема: «СТАНДАРТНА БІБЛІОТЕКА ШАБЛОНІВ (STL) В С++: КОНТЕЙНЕРИ-АДАПТЕРИ, ІТЕРАТОРИ, СТЕК, ЧЕРГА, ЧЕРГА З ПРІОРИТЕТОМ»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: асистент
Козірова Н. Л.

Кропивницький 2025

Мета: ознайомитись з контейнерами-адаптерами, ітераторами, стеком, чергою та чергою з пріоритетом зі стандартної бібліотеки шаблонів (STL) в C++. Навчитись використовувати їх на практиці.

Варіант – 14 - 10 = 4

Завдання:

Розширте попередню систему(ЛР7) **управління складом магазину автозапчастин**, використовуючи асоціативні та адаптивні контейнери STL.

Реалізуйте:

- Збереження запчастин у map, де ключ — це назва запчастини, а значення — структура з виробником, ціною та кількістю.
- Збереження дубльованих запчастин у multimap, якщо в магазині можуть бути однакові запчастини від різних виробників.
- Збереження унікальних назв запчастин у set або multiset.
- Моделювання черги постачання запчастин за допомогою queue.
- Використання priority_queue для черги запчастин з найменшою кількістю на складі (пріоритетне поповнення).
- Використання stack для історії змін даних про запчастини.

Додатково:

Реалізуйте пошук запчастин за виробником або фільтрацію за ціною. Поясніть, як контейнери полегшують організацію даних і які з них доцільні для конкретних задач.

Висновок:

1. **vector<T> –** динамічний масив

Добре підходить, коли потрібен швидкий доступ за індексом і додаються елементи в кінець.

Приклад: список товарів у магазині, де важлива швидка ітерація.

2. **list<T> –** двозв'язний список

Ефективно вставляти/видаляти елементи посередині, але повільний доступ за індексом.

Приклад: черга завдань, де важливе швидке видалення елементів.

3. **queue<T> / stack<T> –** черга і стек

Черга (FIFO) для обробки подій по порядку надходження.

Стек (LIFO) для відкату дій або історії.

Приклад: undo/redo у редакторі → стек; обробка запитів → черга.

4. **set<T> / map<K, V>** – впорядковані множини та словники

set зберігає унікальні значення, автоматично впорядковані.

map дозволяє швидко знаходити значення за ключем.

Приклад: map<string, Part> для швидкого пошуку запчастини за назвою; set<string> для списку унікальних виробників.

5. **multimap<K, V> / multiset<T> –** дозволяють повтори

Використовуються, коли один ключ може мати декілька значень.

Приклад: зберігання всіх постачань одного товару від різних виробників.

Контейнери полегшують код, роблячи його більш зрозумілим і безпечним щодо пам'яті.

Результат:

```
Initial:
* All parts *
Name: Battery, manufacturer: Varta, price: 4200, amount: 15
Name: Brake Pads, manufacturer: Brembo, price: 1250, amount: 40
Name: Oil Filter, manufacturer: Bosch, price: 350, amount: 100
Name: Spark Plug, manufacturer: NGK, price: 270, amount: 80
Name: Summer Tires, manufacturer: Continental, price: 6400, amount: 30
Name: Winter Tires, manufacturer: Michelin, price: 6900, amount: 25

Hendled supply:
* All parts *
Name: Battery, manufacturer: Varta, price: 4200, amount: 15
Name: Brake Pads, manufacturer: Brembo, price: 1250, amount: 50
Name: Oil Filter, manufacturer: Bosch, price: 350, amount: 100
Name: Spark Plug, manufacturer: NGK, price: 270, amount: 80
Name: Summer Tires, manufacturer: Continental, price: 6400, amount: 30
Name: Winter Tires, manufacturer: Michelin, price: 6900, amount: 25
Name: Spark Plug, manufacturer: Denso, price: 270, amount: 20

Enter part name to remove: Oil Filter
Enter manufacturer: Bosch
Enter part name to update: Spark Plug
Enter manufacturer: Denso
Enter new price: 250
Enter new amount: 100

Updated:
* All parts *
Name: Battery, manufacturer: Varta, price: 4200, amount: 15
Name: Brake Pads, manufacturer: Brembo, price: 1250, amount: 50
Name: Spark Plug, manufacturer: NGK, price: 270, amount: 80
Name: Summer Tires, manufacturer: Continental, price: 6400, amount: 30
Name: Winter Tires, manufacturer: Michelin, price: 6900, amount: 25
Name: Spark Plug, manufacturer: Denso, price: 250, amount: 100

Enter manufacturer to search: NGK
* Parts from manufacturer "NGK" *
Name: Spark Plug, manufacturer: NGK, price: 270, amount: 80

Enter min price: 200
Enter max price: 1500
* Parts with price from 200 to 1500 *
Name: Brake Pads, manufacturer: Brembo, price: 1250, amount: 50
Name: Spark Plug, manufacturer: NGK, price: 270, amount: 80
Name: Spark Plug, manufacturer: Denso, price: 250, amount: 100

* Parts History *
Action: updated, name: Spark Plug, manufacturer: Denso, price: 250, amount: 100
Action: removed, name: Oil Filter, manufacturer: Bosch, price: 350, amount: 100
Action: added, name: Spark Plug, manufacturer: Denso, price: 270, amount: 20
Action: added, name: Brake Pads, manufacturer: Brembo, price: 1250, amount: 10
Action: added, name: Summer Tires, manufacturer: Continental, price: 6400, amount: 30
Action: added, name: Winter Tires, manufacturer: Michelin, price: 6900, amount: 25
Action: added, name: Battery, manufacturer: Varta, price: 4200, amount: 15
Action: added, name: Spark Plug, manufacturer: NGK, price: 270, amount: 80
Action: added, name: Oil Filter, manufacturer: Bosch, price: 350, amount: 100
Action: added, name: Brake Pads, manufacturer: Brembo, price: 1250, amount: 40
```

Лістинг Part.h:

```
#ifndef PART_H
#define PART_H

#include <string>
```

```
using namespace std;

struct Part {
    string manufacturer;
    double price = 0.0;
    unsigned short amount = 0;
};

#endif // PART_H
```

Лістинг Supply.h:

```
#ifndef SUPPLY_H
#define SUPPLY_H

#include "Part.h"

struct Supply {
    string name;
    Part part;
};

#endif // SUPPLY_H
```

Лістинг comparator.h:

```
#ifndef COMPARATOR_H
#define COMPARATOR_H

#include "Supply.h"

struct CompareAmount {
    bool operator()(const pair<string, Part>& left, const pair<string, Part>&
right) {
        return left.second.amount > right.second.amount;
    }
};

#endif // COMPARATOR_H
```

Лістинг History.h:

```
#ifndef HISTORY_H
#define HISTORY_H

#include "Part.h"

struct History {
    string action;
    string name;
    Part part;
};

#endif // HISTORY_H
```

Лістинг inventory.h:

```
#ifndef INVENTORY_H
#define INVENTORY_H

#include <map>
```

```
#include <queue>
#include <set>
#include <stack>
#include <string>
#include <iostream>
#include "comparator.h"
#include "History.h"

using namespace std;

class Inventory
{
public:
    Inventory();
    Inventory(const initializer_list<pair<string, Part>>&);

    void addPart(const Supply&);
    void handleSupply();

    template<typename T>
    void showPartsFrom(const T& list) const{
        for (const auto& part : list)
            cout << "Name: " << part.first
                 << ", manufacturer: " << part.second.manufacturer
                 << ", price: " << part.second.price
                 << ", amount: " << part.second.amount << endl;
    }

    void showAllParts() const;

    static void ifFound(bool, const string&, const string&);

    void removePart(const string&, const string&);
    void updatePart(const string&, const string&, double, unsigned short);

    void searchByManufacturer(const string&) const;
    void filterByPrice(double, double) const;

    void showPartsHistory() const;

private:
    map<string, Part> parts;
    multimap<string, Part> duplicates;
    set<string> uniqueNames;
    queue<Supply> supply;

    priority_queue<pair<string, Part>,
                   vector<pair<string, Part>>,
                   CompareAmount> refillQueue;

    stack<History> history;
};

#endif // INVENTORY_H
```

Лістинг inventory.cpp:

```
#include "inventory.h"

Inventory::Inventory() {}
```

```
Inventory::Inventory(const initializer_list<pair<string, Part>>& list) {
    for (const auto& part : list) {
        parts.emplace(part.first, part.second);
        uniqueNames.insert(part.first);

        history.push({"added", part.first, part.second});
    }
}

void Inventory::addPart(const Supply& part) { supply.push(part); }

void Inventory::handleSupply() {
    while (!supply.empty()) {
        Supply sup = supply.front();
        supply.pop();

        auto iterator = parts.find(sup.name);
        if (iterator != parts.end()) {
            if (iterator->second.manufacturer == sup.part.manufacturer)
                iterator->second.amount += sup.part.amount;
            else duplicates.insert({sup.name, sup.part});
        }
        else parts[sup.name] = sup.part;

        history.push({"added" ,sup.name, sup.part});
        uniqueNames.insert(sup.name);
    }
}

void Inventory::showAllParts() const{
    if (parts.empty() && duplicates.empty()) {
        cout << "Inventory is empty(" << endl;
        return;
    }

    cout << "* All parts *" << endl;
    this->showPartsFrom(parts);
    this->showPartsFrom(duplicates);
    cout << endl;
}

void Inventory::ifFound(bool found, const string& name, const string&
manufacturer) {
    if (!found) cout << "Part \"" << name << "\" with manufacturer \"" <<
manufacturer << "\" not found!" << endl;
}

void Inventory::removePart(const string& name, const string& manufacturer) {
    bool found = false;

    auto partIterator = parts.find(name);
    if (partIterator != parts.end() && partIterator->second.manufacturer ==
manufacturer) {
        history.push({"removed", name, partIterator->second});
        parts.erase(partIterator);
        found = true;
    }
```

```
    auto range = duplicates.equal_range(name);
    for (auto dupIterator = range.first; dupIterator != range.second; ) {
        if (dupIterator->second.manufacturer == manufacturer) {
            history.push({"removed", name, dupIterator->second});
            dupIterator = duplicates.erase(dupIterator);
            found = true;
        }
        else ++dupIterator;
    }

    if (parts.find(name) == parts.end() && duplicates.count(name) == 0)
        uniqueNames.erase(name);

    ifFound(found, name, manufacturer);
}

void Inventory::updatePart(const string& name, const string& manufacturer,
double newPrice, unsigned short newAmount) {
    bool found = false;

    auto partIterator = parts.find(name);
    if (partIterator != parts.end() && partIterator->second.manufacturer ==
manufacturer) {
        partIterator->second.price = newPrice;
        partIterator->second.amount = newAmount;
        history.push({"updated", name, partIterator->second});
        found = true;
    }

    auto range = duplicates.equal_range(name);
    auto dupIterator = range.first;
    while (dupIterator != range.second) {
        if (dupIterator->second.manufacturer == manufacturer) {
            dupIterator->second.price = newPrice;
            dupIterator->second.amount = newAmount;
            history.push({"updated", name, dupIterator->second});
            found = true;
        }
        ++dupIterator;
    }

    ifFound(found, name, manufacturer);
}

void Inventory::searchByManufacturer(const string& manufacturer) const{
    map<string, Part> searchedParts;
    multimap<string, Part> searchedDuplicates;

    for (const auto& part : parts)
        if (part.second.manufacturer == manufacturer)
            searchedParts.insert(part);

    for (const auto& part : duplicates)
        if (part.second.manufacturer == manufacturer)
            searchedDuplicates.insert(part);

    if (searchedParts.empty() && searchedDuplicates.empty()) {
        cout << "No parts found from manufacturer \"" << manufacturer <<
"\"." << endl;
```

```
        return;
    }

    cout << "* Parts from manufacturer \"" << manufacturer << "\" *" << endl;
    showPartsFrom(searchedParts);
    showPartsFrom(searchedDuplicates);
    cout << endl;
}

void Inventory::filterByPrice(double minPrice, double maxPrice) const{
    map<string, Part> filteredParts;
    multimap<string, Part> filteredDuplicates;

    for (const auto& part : parts)
        if (part.second.price >= minPrice && part.second.price <= maxPrice)
filteredParts.insert(part);

    for (const auto& part : duplicates)
        if (part.second.price >= minPrice && part.second.price <= maxPrice)
filteredDuplicates.insert(part);

    if (filteredParts.empty() && filteredDuplicates.empty()) {
        cout << "No parts found in this price(" << endl;
        return;
    }

    cout << "* Parts with price from " << minPrice << " to " << maxPrice << "
*" << endl;
    showPartsFrom(filteredParts);
    showPartsFrom(filteredDuplicates);
    cout << endl;
}

void Inventory::showPartsHistory() const {
    if (history.empty()) {
        cout << "History is empty" << endl;
        return;
    }

    cout << "* Parts History *" << endl;
    stack<History> temp = history;

    while (!temp.empty()) {
        const History& note = temp.top();
        cout << "Action: " << note.action
             << ", name: " << note.name
             << ", manufacturer: " << note.part.manufacturer
             << ", price: " << note.part.price
             << ", amount: " << note.part.amount << endl;
        temp.pop();
    }

    cout << endl;
}
```

Лістинг main.cpp:

```
#include "inventory.h"

int main()
```

```
{
    Inventory store = {
        {"Brake Pads", {"Brembo", 1250.0, 40}},
        {"Oil Filter", {"Bosch", 350.0, 100}},
        {"Spark Plug", {"NGK", 270.0, 80}},
        {"Battery", {"Varta", 4200.0, 15}},
        {"Winter Tires", {"Michelin", 6900.0, 25}},
        {"Summer Tires", {"Continental", 6400.0, 30}}
    };

    cout << "Initial:" << endl;
    store.showAllParts();

    store.addPart({"Brake Pads", {"Brembo", 1250.0, 10}});
    store.addPart({"Spark Plug", {"Denso", 270.0, 20}});
    store.handleSupply();

    cout << "Hendled supply:" << endl;
    store.showAllParts();

    string name, manufacturer;
    cout << "Enter part name to remove: ";
    getline(cin, name);
    cout << "Enter manufacturer: ";
    getline(cin, manufacturer);
    store.removePart(name, manufacturer);

    double price = 0.0;
    unsigned short amount = 0;
    cout << "Enter part name to update: ";
    getline(cin, name);
    cout << "Enter manufacturer: ";
    getline(cin, manufacturer);
    cout << "Enter new price: ";
    cin >> price;
    cout << "Enter new amount: ";
    cin >> amount;
    cin.ignore();
    store.updatePart(name, manufacturer, price, amount);

    cout << "\nUpdated: " << endl;
    store.showAllParts();

    cout << "Enter manufacturer to search: ";
    getline(cin, manufacturer);
    store.searchByManufacturer(manufacturer);

    double minPrice, maxPrice;
    cout << "Enter min price: ";
    cin >> minPrice;
    cout << "Enter max price: ";
    cin >> maxPrice;
    store.filterByPrice(minPrice, maxPrice);

    store.showPartsHistory();

    return 0;
}
```