Laborator 11 – Standard Template Library

Tema: Carrefour – Sistem de gestionare a produselor 🛒

Autor: ing. Eduard Donea

Poveste

Carrefour gestionează zilnic mii de produse – alimente, electronice, articole de igienă, toate împărțite pe categorii și căutate de clienți. Angajații au nevoie de un sistem modern, realizat în C++, care să folosească puterea bibliotecii STL pentru a manipula rapid și eficient produsele, coada de clienți, istoricul eliminărilor și statistici automate.

În acest laborator, studenții vor implementa un mic sistem modular **Carrefour.h + Carrefour.cpp** care modelează coșul de cumpărături și operațiunile uzuale folosind containere STL și algoritmi standard.

Scopul laboratorului

- folosirea containerelor STL: `vector`, `list`, `set`, `map`, `queue`, `stack`;
- înțelegerea iteratorilor: normali, inversi, forward;
- utilizarea algoritmilor STL: `sort`, `count`, `max_element`, `lower_bound`;
- folosirea funcțiilor **lambda** ca funcții comparator și predicate;
- structurarea corectă a unui proiect modular;
- folosirea unui **struct** ca model simplu de date (Product);
- procesarea colecțiilor de obiecte în mod eficient și elegant.

Structura proiectului

- **Carrefour.h - Product** – struct ce modelează un produs Carrefour (nume, categorie, preț).
- **Carrefour.h - ShoppingCart** – clasa principală care gestionează toate containerele STL.
- **main.cpp** – fișierul unde sunt testate toate funcționalitățile cerute.

Figura 1 – Diagrama UML a sistemului Carrefour

Fișierul `Carrefour.cpp`

```cpp
#include "Carrefour.h"

// =========================
//      PRODUCT
// =========================

// TODO 0: Implementati constructorul struct-ului Product.

// =========================
//      ADD PRODUCT
// =========================
void ShoppingCart::addProduct(const Product& p) {
    // TODO 1:
    //  - adaugati produsul in vectorul items
    //  - adaugati numele produsului in setul uniqueProducts
    //  - actualizati totalul pe categorie in totalByCategory
}

// =========================
//      REMOVE PRODUCT
// =========================
bool ShoppingCart::removeProduct(const std::string& name) {
    // TODO 2:
    //  1. Folositi std::remove_if cu o lambda pentru a elimina produsele
    //     cu numele "name".
    //  2. Mutati produsele eliminate in removedHistory.
    //  3. Stergeti elementele ramase invalide folosind vector.erase().
    //  4. Returnati true daca s-a sters ceva, false altfel.

    return false; // modificati
}

// =========================
//      SORT LAMBDAS
// =========================
void ShoppingCart::sortByPriceAsc() {
    // TODO 3:
    //  Sortati vectorul items crescator dupa pret folosind
    //  std::sort + lambda.
}

void ShoppingCart::sortByPriceDesc() {
    // TODO 4:
    //  Sortati vectorul items descrescator dupa pret folosind
    //  std::sort + lambda.
}
```

```cpp
void ShoppingCart::sortByName() {
    // TODO 5:
    //  Sortati vectorul items alfabetic dupa nume folosind
    //  std::sort + lambda.
}

// =========================
//      MOST EXPENSIVE
// =========================
Product ShoppingCart::mostExpensive() {
    // TODO 6:
    //  Gasiti produsul cu pretul maxim folosind std::max_element + lambda.
    //  Daca vectorul este gol, returnati Product("None", "None", 0).

    return Product("None", "None", 0); // modificati
}

// =========================
//    COUNT OCCURRENCES
// =========================
int ShoppingCart::countOccurrences(const std::string& name) {
    // TODO 7:
    //  Folositi std::count_if pentru a numara aparitiile unui produs
    //  dupa nume.
}

// =========================
//      FILTER
// =========================
std::vector<Product> ShoppingCart::filterByCategory(const std::string& cat) {
    // TODO 8:
    //  Creati un vector rezultat.
    //  Folositi std::copy_if cu o lambda pentru a adauga doar produsele
    //  din categoria "cat".
    //  Returnati vectorul rezultat.
}

// =========================
//   PRINT WITH ITERATORS
// =========================
void ShoppingCart::printWithIterators() {
    // TODO 9:
    //  Parcurgeti vectorul folosind un iterator explicit (begin() -> end()).
}

void ShoppingCart::printWithReverseIterators() {
    // TODO 10:
    //  Parcurgeti vectorul in ordine inversa folosind un reverse_iterator.
}
```

```cpp
// =========================
//         QUEUE
// =========================
void ShoppingCart::addCustomer(const std::string& c) {
    // TODO 11:
    //  Adaugati clientul in coada (queue).
}

void ShoppingCart::serveCustomer() {
    // TODO 12:
    //  - Daca nu sunt clienti, afisati un mesaj.
    //  - Altfel, afisati clientul servit si scoateti-l din coada.
}

// =========================
//         STACK
// =========================
void ShoppingCart::addBag(const std::string& b) {
    // TODO 13:
    //  Adaugati numele unei plase in stiva (stack).
}

void ShoppingCart::useBag() {
    // TODO 14:
    //  - Daca nu mai sunt plase, afisati un mesaj.
    //  - Altfel, afisati plasa folosita si scoateti-o din stiva.
}

// =========================
//         PRINT CART
// =========================
void ShoppingCart::printCart() const {
    // TODO 15:
    //  Afisati toate produsele din cos (items).
}

void ShoppingCart::printRemovedHistory() const {
    // TODO 16:
    //  Afisati toate produsele eliminate (removedHistory).
}
```

Struct-ul `Product`

Struct-ul `Product` reprezintă un tip de date simplu, optim pentru a stoca doar informație, fără comportament complex.
În C++, un struct este echivalent cu o clasă cu membri publici, iar în UML este reprezentat identic cu o clasă, deoarece este parte din modelul de date al aplicației.

Clasa `ShoppingCart`

Clasa `ShoppingCart` este centrul aplicației Carrefour. Ea utilizează multiple containere STL:

- `std::vector<Product>| - lista principală de produse;`
- `std::list<Product>| - istoricul eliminărilor;`
- `std::set<std::string>| - evidența numelor unice;`
- `std::map<std::string, float>| - totaluri pe categorii;`
- `std::queue<std::string>| - coada de clienți;`
- `std::stack<std::string>| - plasele de cumpărături.`

Clasa va integra algoritmi STL precum sortare, căutare binară, numărare, determinarea maximului, precum și funcții lambda pentru comparatori și predicate de filtrare.

Fișierul `main.cpp`

```cpp
#include "Carrefour.h"

int main() {

    ShoppingCart cart;

    // ================================
    // TODO A: Adaugati cateva produse
    // ================================
    // Exemplu:
    // cart.addProduct({"Lapte", "Lactate", 7.5});
    // Adaugati minim 5 produse in cos.

    // TODO B: Afisati cosul

    // ================================
    // TODO C: Sortari cu lambda
    // ================================
    // 1) sortByPriceAsc()
    // 2) sortByPriceDesc()
    // 3) sortByName()
    //
    // Apelati sortarile si afisati cosul dupa fiecare.

    // ================================
    // TODO D: Gasiti produsul cel mai scump
    // ================================

    // ================================
    // TODO E: Stergere produse cu remove_if
    // ================================
    // Incercati sa stergeti un produs (ex: "Paine")
    // Afisati si istoricul produselor sterse.

    // ================================
    // TODO F: Filtrare cu copy_if
    // ================================
    // Creati un vector cu toate produsele dintr-o categorie,
    // ex: "Lactate" si afisati-l.

    // ================================
    // TODO G: Iteratori si reverse iteratori
    // ================================
```

```
    // ================================
    // TODO H: Coada de clienti (queue)
    // ================================
    // Adaugati 3 clienti la coada
    // Exemplu:
    // cart.addCustomer("Ana");
    //
    // Serviti macar 2 clienti folosind serveCustomer();

    // ================================
    // TODO I: Stiva de plase (stack)
    // ================================
    // Adaugati 2-3 plase in stiva si folositi cateva.

    // ================================
    // TODO J: Lambda capturi
    // ================================
    // Creati un vector filtrat si aplicati o reducere de pret
    // folosind std::for_each + lambda cu capturi.
    //
    // Exemplu:
    //
    // int discount = 10;
    // int ops = 0;
    //
    // std::for_each(result.begin(), result.end(),
    //     [=, &ops](Product p) mutable {
    //         float newPrice = p.price - discount;
    //         ops++;
    //         std::cout << p.name << " -> " << newPrice << " lei\n";
    //     });
    //
    // std::cout << "Operatii lambda: " << ops << "\n";


    return 0;
}
```

Cerințe (10p)

1. (1p) Implementați struct-ul `Product`.
2. (4p) Implementați clasa `ShoppingCart` cu toate containerele cerute.
3. (2p) Realizați sortări folosind lambda functions (după nume și preț).
4. (1p) Implementați metoda de filtrare după categorie.
5. (2p) Implementați în `main.cpp` toate operațiile cerute în TODO.

Compilare și rulare

```
g++ -std=c++17 -g *.cpp -o lab11
valgrind ./lab11
```

Dacă programul nu compilează, se vor scădea 4p.

Bonus GitHub Classroom

Studentii pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului sau **1p** dacă trimit cel puțin jumătate.

1. Accesați cursul de pe OCW – Programare Orientată pe Obiecte.
2. Mergeți la secțiunea **Bonus GitHub laborator**.
3. Acceptați tema pentru Laborator 11.

Atenție: Repository-urile sunt monitorizate automat. Codul copiat între studenți anulează bonusul pentru toți participanții.