Laborator 11

Tema: Standard Template Library (STL) 🍴

Poveste

O platformă de livrări mâncare dorește să optimizeze modul în care gestionează meniurile restaurantelor partenere. Fiecare restaurant are o listă de preparate (**Dish**), iar fiecare preparat are un nume, un preț și o listă de ingrediente. Platforma are nevoie de funcționalități avansate de analiză și filtrare pentru a îmbunătăți experiența utilizatorilor:

- Sortarea preparatelor după preț pentru clienții cu buget redus.
- Căutarea rapidă a preparatelor care conțin sau nu anumite ingrediente (pentru clienții cu alergii sau preferințe dietetice).
- Generarea de statistici privind cele mai folosite ingrediente în meniu.

Deoarece cerințele se pot schimba rapid și volumul de date este variabil, vom renunța la tablourile statice și gestionarea manuală a memoriei în favoarea **Standard Template Library (STL)**.

Scopul laboratorului

- Utilizarea containerelor secvențiale (`std::vector`, `std::list`);
- Utilizarea containerelor asociative (`std::map`) pentru statistici;
- Folosirea clasei `std::string` pentru manipularea textului;
- Scrierea de funcții **Lambda** pentru criterii custom de sortare și filtrare;
- Utilizarea algoritmilor standard: `sort`, `find_if`, `count_if`.

Cerințe de implementare (9p + 1p oficiu)

Deoarece lucrăm cu STL, în acest laborator **vom folosi** `std::string` în loc de `char*`.

1. **Clasa Dish (2p)**
   - Creați o clasă simplă `Dish` care să conțină:
     - Un nume (`std::string`);
     - Un preț (`double`);
     - O listă de ingrediente (folosiți `std::vector<std::string>`).
   - Implementați constructorul și, opțional, operatorul `«` pentru afișare.
2. **Gestiunea Meniului (3p)**
   - În `main`, populați un `std::vector<Dish>` cu cel puțin 5 preparate diverse.
   - Folosind algoritmul `std::sort` și o **funcție lambda**, sortați meniul crescător după preț și afișați-l.
   - Sortați din nou meniul alfabetic după nume, folosind o altă expresie lambda.
3. **Căutare și Filtrare (2p)**
   - Folosind algoritmul `std::find_if`, căutați un preparat specific după nume (citit de la tastatură sau hardcodat).
   - Implementați o filtrare care afișează doar preparatele care **NU** conțin un anumit ingredient (ex: "ciuperci"). Puteți folosi o buclă *range-based for* sau algoritmul `copy_if`.
4. **Statistici cu Map (2p)**
   - Folosiți un `std::map<std::string, int>` pentru a număra de câte ori apare fiecare ingredient în întregul meniu.
   - Cheia va fi numele ingredientului, iar valoarea va fi numărul de apariții.
   - Iterați prin map și afișați ingredientele și frecvența lor.

Ghid pentru `main.cpp`

Acesta este singurul cod pe care îl primiți. Trebuie să creați fișierele `Dish.h` și `Dish.cpp` și să completați `main`-ul conform comentariilor.

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm> // Pentru sort, find_if
#include <map>       // Pentru statistici
#include "Dish.h"    // Trebuie sa creati voi acest fisier
int main() {
    // 1. Initializare meniu (vector de obiecte Dish)
    std::vector<Dish> meniu;

    // TODO: Adaugati 5 preparate folosind push_back.
    // Fiecare preparat are nume, pret, si vector de ingrediente.
    // Ex: meniu.push_back("Pizza", 45.5, std::vector<std::string>{"faina",
    ↪  "rosii"});

    // 2. Afisare initiala
    std::cout << "--- Meniu Initial ---\n";
    for (const auto& d : meniu) {
        // TODO: Afisati preparatul (supraincarcati operator << in Dish sau metoda
        ↪  print)
    }

    // 3. Sortare dupa pret (Lambda)
    // TODO: Folositi std::sort cu o functie lambda care compara preturile a doua
    ↪  obiecte Dish.
    // std::sort(meniu.begin(), meniu.end(), [](const Dish& a, const Dish& b) { ...
    ↪  });

    std::cout << "\n--- Meniu Sortat dupa Pret ---\n";
    // TODO: Afisare meniu sortat

    // 4. Cautare preparat dupa nume
    std::string cautat = "Pizza";
    // TODO: Folositi std::find_if cu o lambda care verifica daca numele preparatului
    ↪  este egal cu 'cautat'.
    // find_if returneaza un iterator. Verificati daca iteratorul != meniu.end().

    // 5. Statistici ingrediente (std::map)
    std::map<std::string, int> ingredienteCount;
    // TODO: Iterati prin fiecare Dish din meniu.
    // TODO: In interior, iterati prin vectorul de ingrediente al acelui Dish.
    // TODO: Incrementati valoarea in map: ingredienteCount[ingredient]++;

    std::cout << "\n--- Top Ingrediente ---\n";
    // TODO: Iterati prin map si afisati ingredientul si numarul de utilizari.
    return 0;
}
```

Compilare și verificare

```
# Compilare simpla (daca implementati Dish in .h si .cpp)
g++ -o lab11 main.cpp Dish.cpp
# Rulare
./lab11
# Verificare memory leaks (optional, dar recomandat)
valgrind ./lab11
```

Bonus GitHub Classroom

Studenții pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului.

Pași pentru încărcarea laboratorului:

(a) Accesați linkul **GitHub Classroom** pentru Laboratorul 11:

`https://classroom.github.com/a/RpAZgwml`

(b) Clonați repository-ul, creați fișierele necesare (`Dish.h`, `Dish.cpp`) și completați `main.cpp`.

(c) Încărcați soluția folosind comenzile git:

```
git add .
git commit -m "Laborator 11 - STL Implementation"
git push origin main
```