Ministerul Educației și Cercetării al Republicii Moldova
Centrul de Excelență în Informatică și Tehnologii Informaționale
Catedra Informatica II

Raport

pentru practica de inițiere în specialitate

Specialitatea: Programarea și testarea produselor program

Tema: Băuturi Răcoritoare

Realizat - Pricop Maxim P-2413

Verificat - Gairunova Natalia

Nota -

2025

Cuprins

Introducere

Acest raport va prezenta activităţile şi rezultatele stagiului de practică de iniţiere în specialitate, în cadrul căruia s-a creat un sistem simplu pentru gestionarea băuturilor răcoritoare şi a livrărilor asociate cu ele. În paginile următoare se va urmări consolidarea cunoştinţelor teoretice şi aprofundarea deprinderilor practice dobândite la modulele de programare structurată, programare procedurală și procesarea informaţiei, se vor aplica tehnologii de elaborare a produselor program și de creare a registrelor de calcul tabelar.

Programul C++ se bazează pe fişierele text *Baut.txt* (stocându-se ID-ul, tipul, denumirea, culoarea şi preţul pe litru al fiecărei băuturi) şi *Livr.txt* (conţinând ID-ul livrării, ID-ul produsului şi cantitatea livrată). Programul lucrează în consolă printr-un meniu intuitiv ce oferă utilizatorului posibilitatea adăugării, modificării, ştergerii şi afişării rapide a înregistrărilor despre produse şi livrări.

Componenta Microsoft Excel preia automat datele din fişierele text şi le transpune în foi de calcul structurate. Se definesc liste derulante pentru validarea informaţiilor şi se utilizează formule automate pentru calculul stocurilor disponibile şi al valorii livrărilor, iar graficele generate oferă o perspectivă vizuală asupra evoluţiei volumelor şi a veniturilor.

Sarcini

Sarcini C++

În cadrul modulului C++ s-a realizat o aplicație de consolă care gestionează înregistrările despre băuturi răcoritoare și livrări, lucrând direct cu fișierele text. Programul oferă un meniu simplu, unde utilizatorul poate vizualiza datele curente, le poate adăuga sau modifica, și generează rapoarte sumare.

Lista sarcinilor în C++:

- citirea și afișarea completă a conținutului fișierelor *Baut.txt* și *Livr.txt* în consolă;
- gestionarea prin funcții modulare a operațiilor de adăugare, ștergere și actualizare a înregistrărilor de băuturi și de livrări, cu verificarea prealabilă a datelor introduse;
- generarea automată a fișierului RezumatLivrari.txt, care acumulează pentru fiecare băutură cantitatea totală livrată și valoarea aferentă (cantitate × preț);
- filtrarea și ordonarea listelor după criterii precum tipul băuturii sau denumire, plus identificarea produselor cu cele mai mari și cele mai mici valori totale de livrare;
- calcularea statisticilor necesare (valoarea maximă/minimă a livrărilor şi prețul mediu pe litru pentru un prag de cantitate specificat de utilizator).

Sarcini MS Excel

În Excel s-a creat un registru de calcul tabelar care importă direct datele din fișierele text și le transformă în tabele structurate, folosind validări, formule și elemente vizuale pentru o analiză rapidă a datelor.

Lista sarcinilor în MS Excel:

- importarea automată a datelor din *Baut.txt* şi *Livr.txt* în două foi separate denumite „Bauturi” şi „Livrări”, cu formatarea corespunzătoare a tabelelor;
- configurarea listelor derulante pentru câmpurile critice (tip, denumire, culoare) și adăugarea unei coloane de imagini pentru fiecare băutură, precum și inserarea unui comentariu ilustrativ;
- introducerea formulelor care calculează stocurile rămase şi „Suma achitată” în euro (cu conversie după curs), plus un rând de totaluri pentru fiecare tabel;
- realizarea foii „Statistica”, unde se agregă numărul de livrări, cantitatea totală și suma încasată pentru băutura aleasă de utilizator, și crearea de grafice dinamice care reflectă evoluția volumelor și a veniturilor;
- implementarea unui ComboBox interactiv pentru selectarea băuturii (ordonată alfabetic) și afișarea automată a imaginii produsului selectat.

Funcționalitatea programului

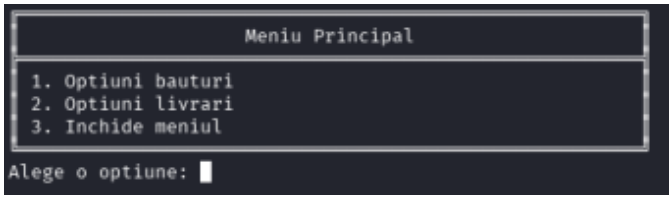

Figura 1

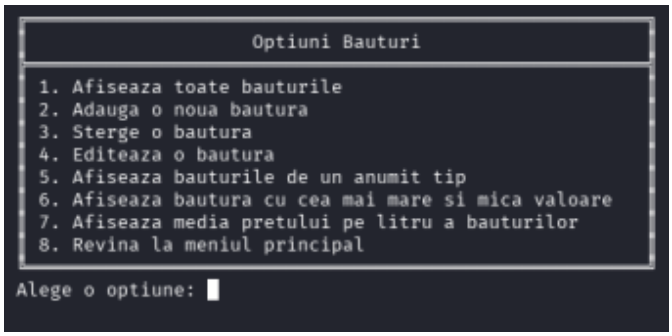

Figura 2

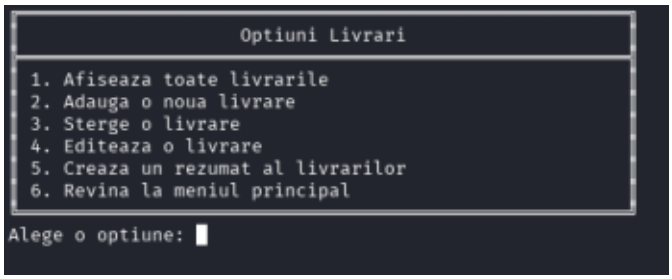

Figura 3

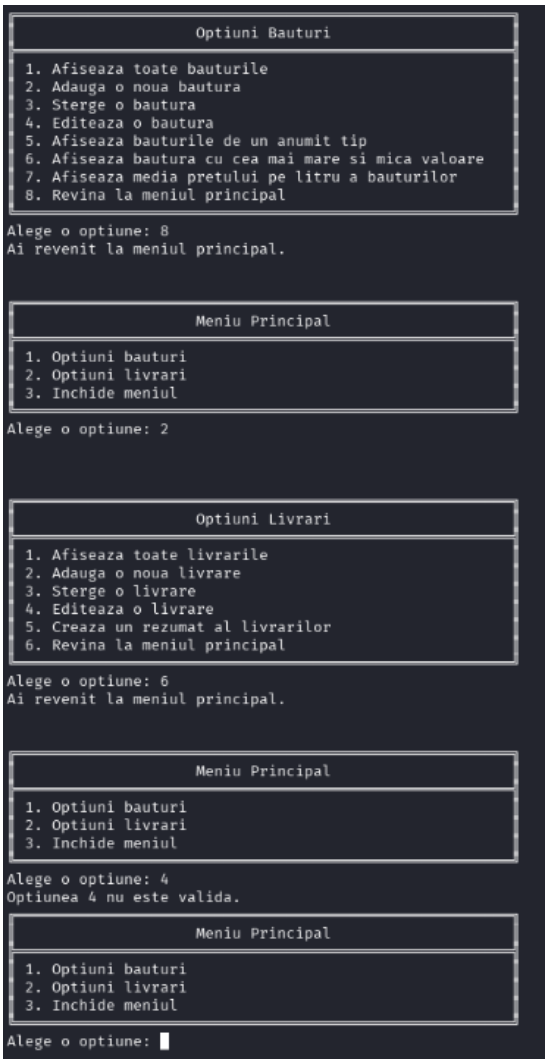

Figura 4

În figurile 1, 2, 3 și 4 se poate vedea funcționalitatea meniurilor. Opțiunile de livrări și de băuturi sunt separate în meniuri diferite, schimbare între meniuri este posibilă fără a fi nevoie de a reporni programul, iar el nu se va închide până când utilizatorul nu va selecta opțiunea de închidere datorită ajutorului unei bucle infinite.

Am ales să împart meniul principal în 2 meniuri mai mici pentru îmbunătățirea experienței utilizatorului și micșorarea meniului principal.

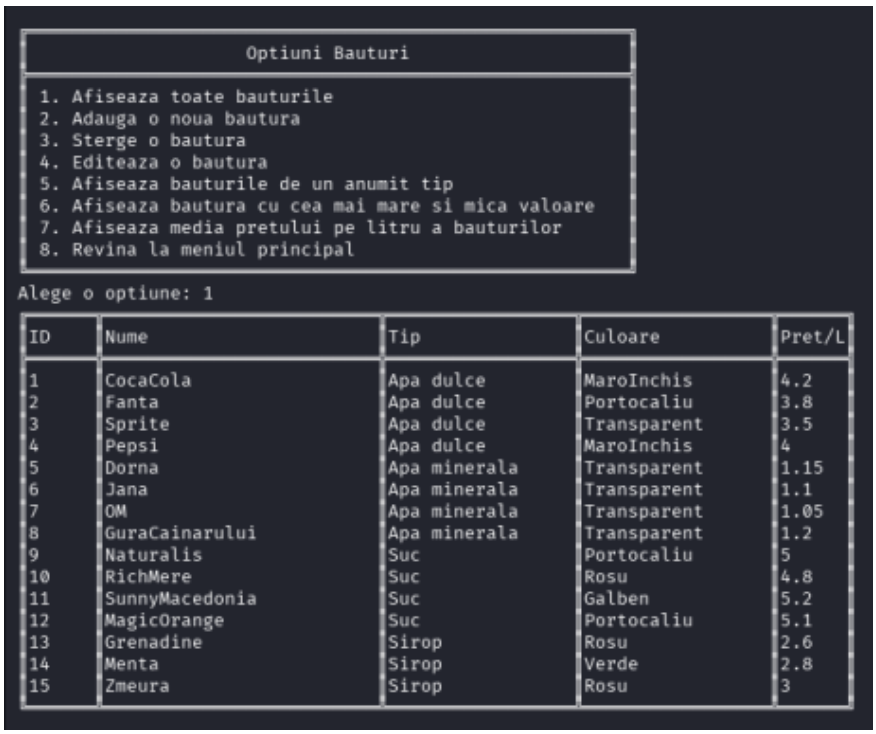

Figura 5

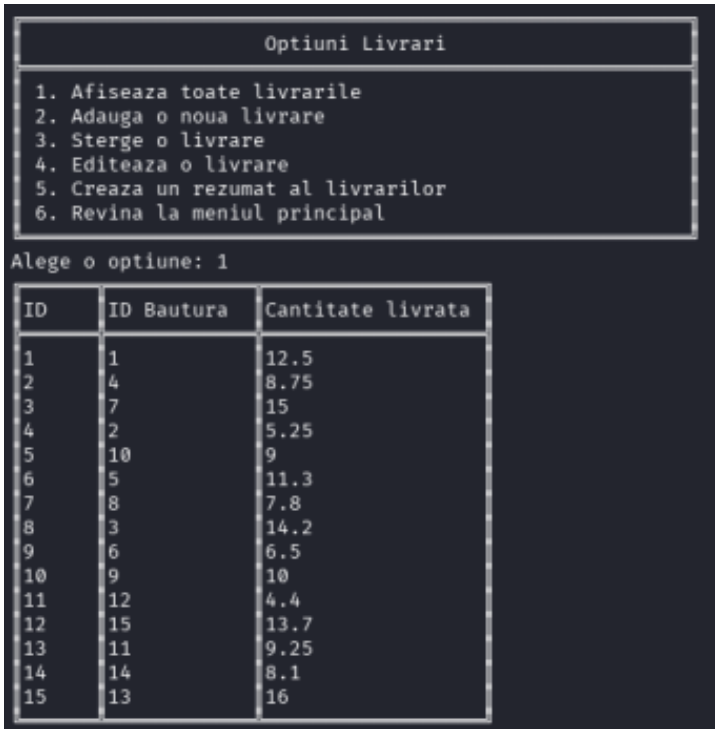

Figura 6

Figurile 5 și 6 arată funcționalitatea de a afișa în consolă a tuturor băuturilor și a livrărilor din fișierele text.

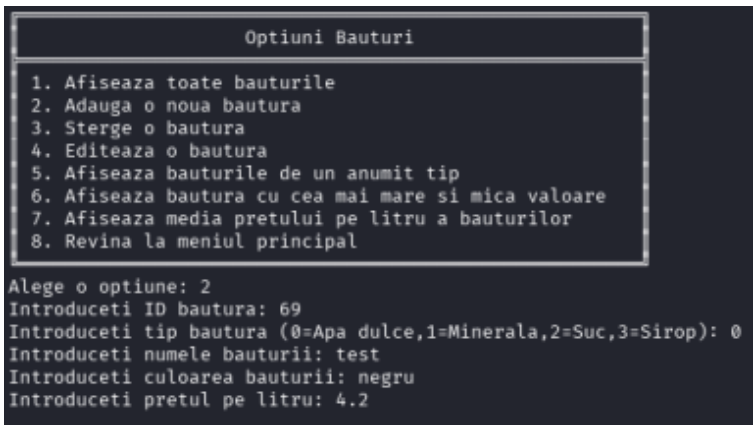

Figura 7

Figurile 7 și 8 arată funcționalitatea opțiunilor de ștergere și de adăugare a unei noi date. Ștergerea datelor funcționează în bază de ID pentru livrări și în baza numelui pentru băuturi.

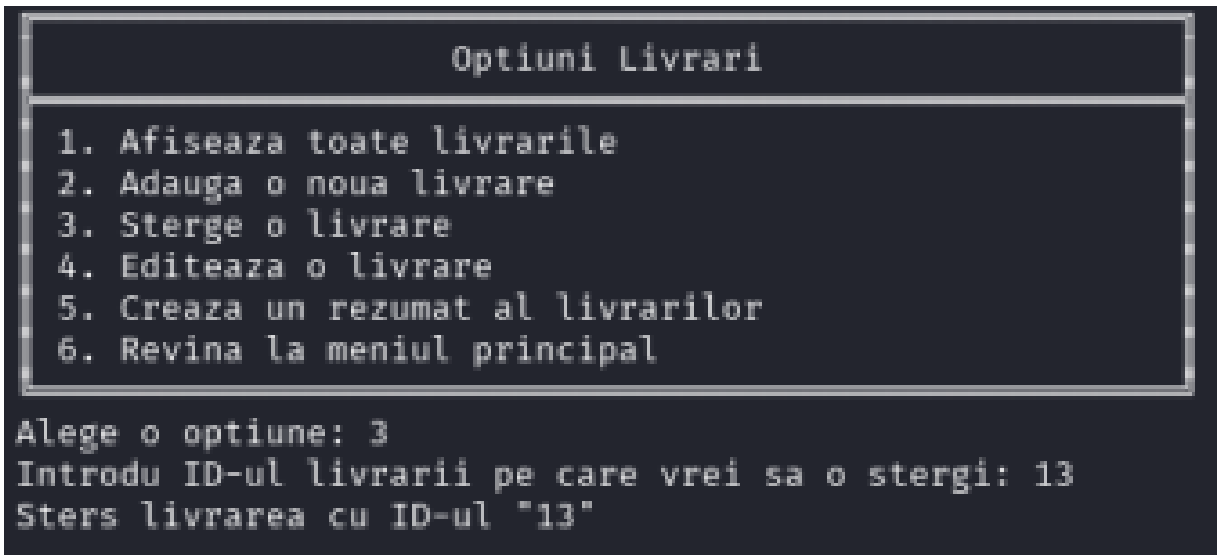

Figura 8

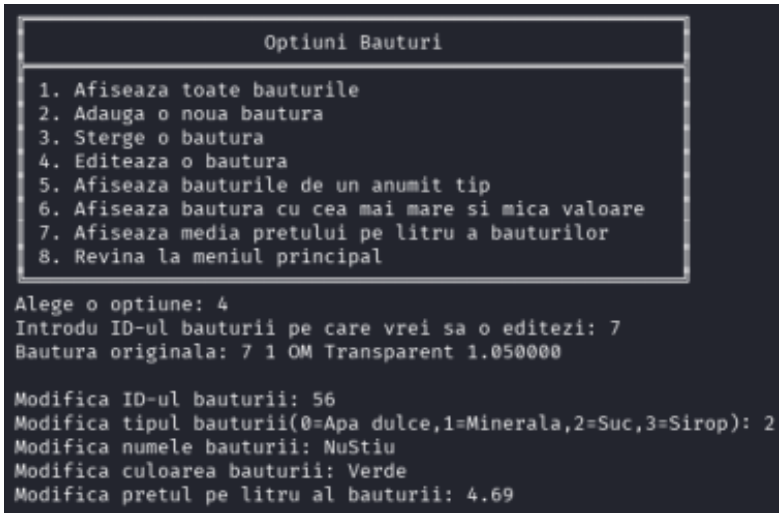

Figura 9

În figura 9 este reprezentat editarea unei băuturi. Înainte de editare, utilizatorul poate vedea cum intrarea arăta inițial și apoi poate modifica orice aspect al acelei intrări.

Figura 10

Figura 11

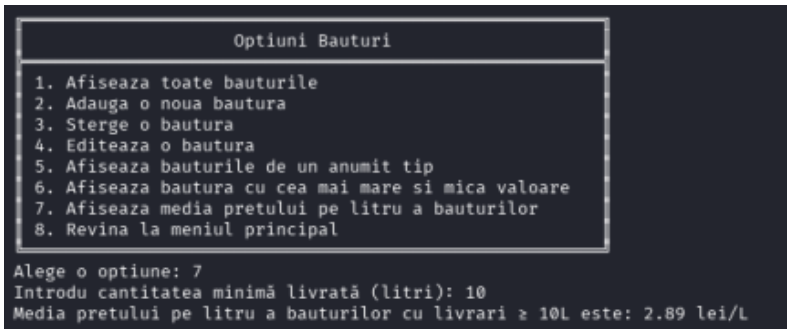

Figura 12

În figurile 10, 11 și 12 sunt arătate ultimele opțiuni a băuturilor. În figura 10 sunt arătate băuturile doar de un anumit tip selectat de utilizator în ordine alfabetică. În figura 11 sunt arătate băturile cu cea mai mare și cea mai mică valoare, iar în figura 12 este arătat pretul mediu pe litru pentru băuturile ce au livrări cu o cantitate minimă introdusă de utilizator.

Codul sursă C++

În acest capitol se va prezenta structura internă a codului, modul în care componentele interacţionează pentru a îndeplini cerinţele aplicaţiei şi raţionamentul din spatele alegerilor de implementare, evidenţiind avantajele abordării adoptate în comparaţie cu soluţii alternative.

Structura fișierelor

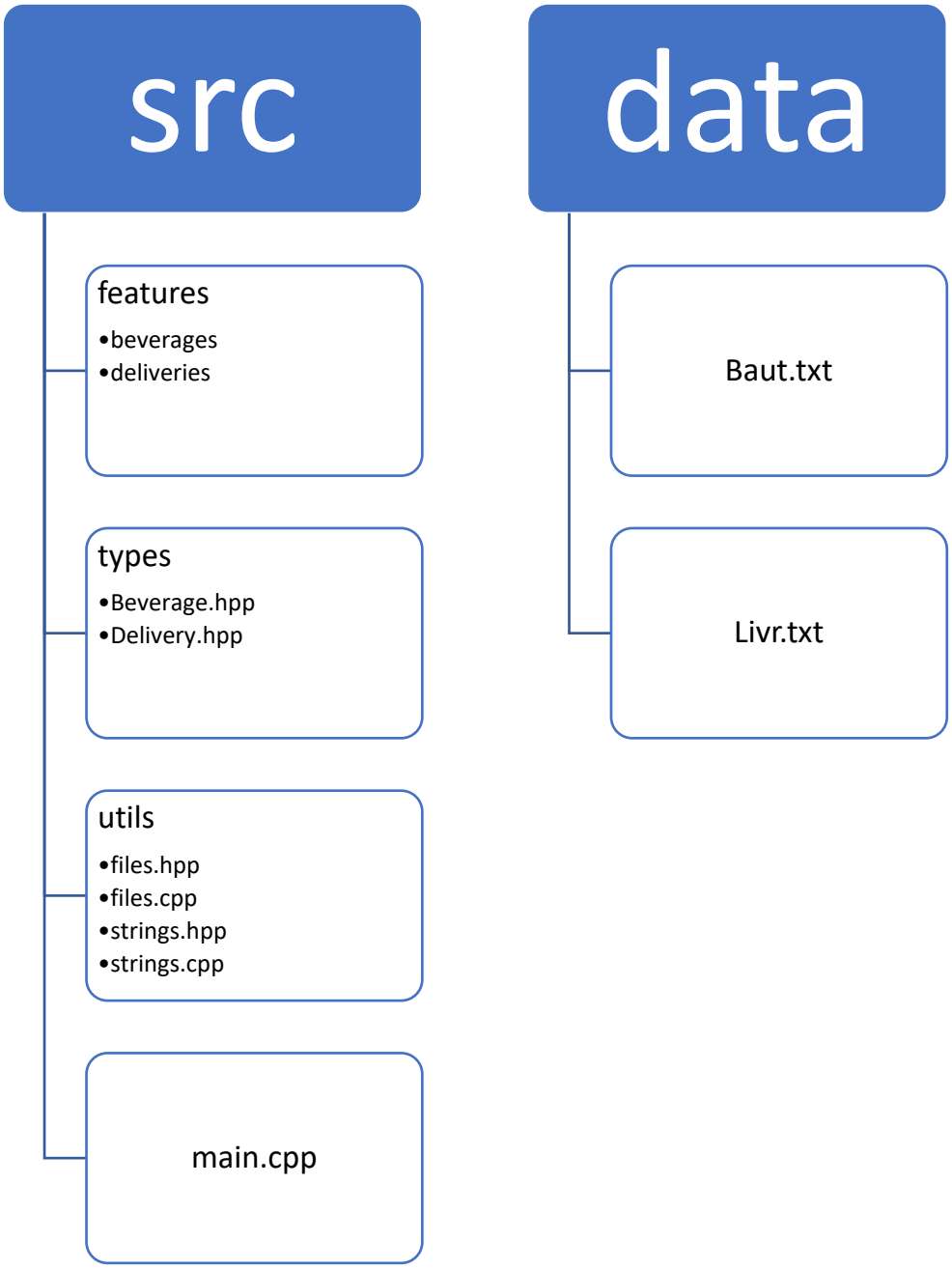

Figura 13

Mai sus, în figura 13, se poate vedea o ilustrație grafică a structurii programului. El este împărțit în 2 dosare. Primul este numit data și conține cele 2 fișiere de text cu informațiile despre băuturi și livrări. Al doilea este numit src și conține tot codul sursă. Dosarul src are în el alte 3 dosare, dar și fișierul *main.cpp*.

Primul dosar este numit types, ce conține 2 fișiere header de tipul .hpp cu tipurile pentru livrări și băuturi ce sunt folosite în tot codul.

Al doilea dosar, este numit utils și conține diferite funcții ajutătoare ce sunt folosite în mai multe fișiere și locuri diferite. El conține 2 fișiere, primul fișier numit *files.cpp* conține funcții ce ajută la manipularea fișierelor text, iar al doilea fișier numit *strings.cpp* conține funcții ce ajută la manipularea variabilelor de tip string.

Al treilea dosar numit features are alte 2 dosare, unul numit *beverages* și celelalt numit *deliveries*. Aceste dosare conțin funcțiile principale pentru lucrul cu băuturile și livrările.

Explicarea codului

În această secțiune se va explică codul și raționamentul din spatele alegerilor de implementare a lui. Mai jos sunt doar câteva secvențe de cod. Codul complet poate fi găsit pe linkurile din anexa 1 și 2.

Tipurile create

Pentru a stoca informațiile eficient și a facea lucrarea cu ele mult mai ușoară în cod, am avut nevoie să creez 3 tipuri:

- Delivery – Acest tip conține toate informațiile despre o singură livrare
- BeverageType – Acest tip este creat folosin enum și conține cele 4 opțiuni de tipuri de băuturi posibile. (Apă dulce, minerală, suc, sirop)
- Beverage – Acest tip contține toate informațiile despre o singură băutură.

Fișierul Delivery.hpp:

```
#pragma once

struct Delivery {
    unsigned id;
    unsigned beverageId;
    double quantityDelivered;
};
```

Fișierul Beverages.hpp:

```
#pragma once
#include "string"

enum BeverageType {
    Sweet,
    Mineral,
    Juice,
    Syrup
};

struct Beverage {
    unsigned id;
    BeverageType type;
    std::string name;
    std::string color;
    float pricePerLiter;
};
```

Fișierul main

Fișierul main conține 3 funcții. Prima funcție, este acea main ce are un meniu principal cu 3 opțiuni – accesarea opțiuni băuturi, accesarea opțiuni livrări și închiderea meniului. Tot meniul este într-o buclă while infinită pentru a preveni oprirea programului când utilizatorul selectează o opțiune. Așa el va rula la infinit și programul se va opri doar dacă utilizator alege această opțiune. În același timp, previne și alegerea unei opțiuni invalide.

```
int main() {
    while (true) {
        short unsigned userChoice;

        std::cout <<
"╔═══════════════════════════════════════════════════════════╗\n";
        std::cout << "║                          Meniu
Principal                        ║\n";
        std::cout <<
"╠═══════════════════════════════════════════════════════════╣\n";
        std::cout << "║ 1. Optiuni
bauturi                                   ║\n";
        std::cout << "║ 2. Optiuni
livrari                                   ║\n";
        std::cout << "║ 3. Inchide
meniul                                    ║\n";
        std::cout <<
"╚═══════════════════════════════════════════════════════════╝\n";
        std::cout << "Alege o optiune: ";
        std::cin >> userChoice;

        switch (userChoice) {
            case 1: {
                std::cout << "\n\n\n";
                beverageOptions();

                break;
            }
            case 2: {
                std::cout << "\n\n\n";
                deliveryOptions();

                break;
            }
            case 3: {
                std::cout << "Ai inchis meniul.";
                return 0;
            }
            default: {
                std::cout << "Optiunea " << userChoice << " nu este valida.\n";
                break;
            }
        }
    }

    return 0;
}
```

Alegerea opțiunii 1 sau 2, te redirecționează la una din celelalte 2 funcții ce conțiune meniuri de menajare a băuturilor sau a livrărilor. Aceste funcții au și ele câte un meniu pentru selectarea acțiunii dorite de utilizator. Folosin switch case, ele redirecționează la alte funcții din dosarul *features* ce conțin codul acțiunii dorite de utilizator.

```
void deliveryOptions() {
    short unsigned userChoice;

    std::cout << "╔═══════════════════════════════════════════════════╗\n";
    std::cout << "║                  Optiuni Livrari                  ║\n";
    std::cout << "╠═══════════════════════════════════════════════════╣\n";
    std::cout << "║ 1. Afiseaza toate livrarile                       ║\n";
    std::cout << "║ 2. Adauga o noua livrare                          ║\n";
    std::cout << "║ 3. Sterge o livrare                               ║\n";
    std::cout << "║ 4. Editeaza o livrare                             ║\n";
    std::cout << "║ 5. Creaza un rezumat al livrarilor                ║\n";
    std::cout << "║ 6. Revina la meniul principal                     ║\n";
    std::cout << "╚═══════════════════════════════════════════════════╝\n";
    std::cout << "Alege o optiune: ";
    std::cin >> userChoice;

    switch (userChoice) {
        case 1: {
            displayDeliveries("../data/Livr.txt");
            break;
        }
        case 2: {
            addNewDelivery("../data/Livr.txt");
            break;
        }
        case 3: {
            deleteDelivery("../data/Livr.txt");
            break;
        }
        case 4: {
            editDelivery("../data/Livr.txt");
            break;
        }
        case 5: {
            summarizeDeliveries("../data/Baut.txt", "../data/Livr.txt",
"../data/RezumatLivrari.txt");
            break;
        }
        case 6: {
            std::cout << "Ai revenit la meniul principal.";
            break;
        }
        default: {
            std::cout << "Optiunea " << userChoice << " nu este valida.";
            break;
        }
    }

    std::cout << "\n\n\n";
}
```

```
void beverageOptions() {
    short unsigned userChoice;

    std::cout << "╔═══════════════════════════════════════════════════════╗\n";
    std::cout << "║                    Optiuni Bauturi                    ║\n";
    std::cout << "╠═══════════════════════════════════════════════════════╣\n";
    std::cout << "║ 1. Afiseaza toate bauturile                           ║\n";
    std::cout << "║ 2. Adauga o noua bautura                              ║\n";
    std::cout << "║ 3. Sterge o bautura                                   ║\n";
    std::cout << "║ 4. Editeaza o bautura                                 ║\n";
    std::cout << "║ 5. Afiseaza bauturile de un anumit tip                ║\n";
    std::cout << "║ 6. Afiseaza bautura cu cea mai mare si mica valoare   ║\n";
    std::cout << "║ 7. Afiseaza media pretului pe litru a bauturilor      ║\n";
    std::cout << "║ 8. Revina la meniul principal                         ║\n";
    std::cout << "╚═══════════════════════════════════════════════════════╝\n";
    std::cout << "Alege o optiune: ";
    std::cin >> userChoice;

    switch (userChoice) {
        case 1: {
            displayBeverages("../data/Baut.txt");
            break;
        }
        case 2: {
            addNewBeverage("../data/Baut.txt");
            break;
        }
        case 3: {
            deleteBeverage("../data/Baut.txt");
            break;
        }
        case 4: {
            editBeverage("../data/Baut.txt");
            break;
        }
        case 5: {
            displaySpecificBeverage("../data/Baut.txt");
            break;
        }
        case 6: {
            displayBeverageValueExtremes("../data/Baut.txt",
"../data/Livr.txt");
            break;
        }
        case 7: {
            displayAveragePriceForQuantity("../data/Baut.txt",
"../data/Livr.txt");
            break;
        }
        case 8: {
            std::cout << "Ai revenit la meniul principal.";
            break;
        }
        default: {
            std::cout << "Optiunea " << userChoice << " nu este valida.";
            break;
        }
    }

    std::cout << "\n\n\n";
}
```

Funcțiile utils

Funcțiile din dosarul *utils* sunt funcții ajuătoare ce sunt folosite în mai multe fișiere pentru diferite scopuri. Ele sunt împărțite în funcții ajuătoare pentru fișiere și pentru strings. În total sunt 4 fișiere, 2 fișiere header și 2 fișiere C++ ce conține codul.

În total sunt 5 funcții ajuătoare pentru fișiere. Câte o funcție pentru citirea băuturilor și a livrărilor dintr-un fișier specificat, scrierea băuturilor și a livrărilor într-un fișier specificat și cititea doar a unui tip specificat de băuturi dintr-un fișier.

files.hpp

```
#pragma once
#include "../types/Beverages.hpp"
#include "../types/Delivery.hpp"
#include "string"
#include "vector"

std::vector<Beverage> getAllBeverages(const std::string &filename);

std::vector<Beverage> getSpecificBeverage(const std::string &filename,
BeverageType &type);

void writeBeverages(const std::vector<Beverage> &beverages, const std::string
&filename);

std::vector<Delivery> getAllDeliveries(const std::string &filename);

void writeDeliveries(const std::vector<Delivery> &deliveries, const std::string
&filename);
```

Funcțiile de citire a bauturilor și a livrărilor sunt similare între ele. Ele folosect un vector pentru a stoca toate băuturile/livrările, iar la final ele întorc acel vector. Folosind o buclă while, citesct până la finalul fișierului toate informațiile despre livrare/băuturi.

```
std::vector<Beverage> getAllBeverages(const std::string &filename) {
    std::vector<Beverage> beverages;
    std::ifstream file(filename);

    if (!file.is_open()) return beverages;

    Beverage currentBeverage;

    // Expected line layout:
    // ID (unsigned), Type (BeverageType), name (string), color (string),
pricePerLiter (float)
    while (file >> currentBeverage.id) {
        int rawBeverageType;
        file >> rawBeverageType >> currentBeverage.name >> currentBeverage.color
>> currentBeverage.pricePerLiter;

        currentBeverage.type = static_cast<BeverageType>(rawBeverageType);
        beverages.push_back(currentBeverage);
    }

    file.close();
    return beverages;
}
```

```
std::vector<Delivery> getAllDeliveries(const std::string &filename) {
    std::vector<Delivery> deliveries;
    std::ifstream file(filename);

    if (!file.is_open()) return deliveries;

    Delivery currentDelivery;

    // Expected line layout:
    // ID (unsigned), Beverage ID (unsigned), Quantity Delivered (double)
    while (file >> currentDelivery.id) {
        file >> currentDelivery.beverageId >> currentDelivery.quantityDelivered;
        deliveries.push_back(currentDelivery);
    }

    file.close();
    return deliveries;
}
```

Funcția de citire a băuturilor în baza tipului lor este aproape indetincă cu cea de citire a tuturor băuturilor, doar că introduce o verificare a tipului după citirea informațiilor din fișierul text. Dacă este de tipul corect, atunci este adăugată la vectorul final, în caz că nu, se continuă mai departe.

```
std::vector<Beverage> getSpecificBeverage(const std::string &filename,
BeverageType &type) {
    std::vector<Beverage> beverages;
    std::ifstream file(filename);

    if (!file.is_open()) return beverages;

    Beverage currentBeverage;

    // Expected line layout:
    // ID (unsigned), Type (BeverageType), name (string), color (string),
pricePerLiter (float)
    while (file >> currentBeverage.id) {
        int rawBeverageType;
        file >> rawBeverageType >> currentBeverage.name >> currentBeverage.color
>> currentBeverage.pricePerLiter;

        currentBeverage.type = static_cast<BeverageType>(rawBeverageType);
        if (currentBeverage.type != type) continue;

        beverages.push_back(currentBeverage);
    }

    file.close();
    return beverages;
}
```

Funcțiile de scriere a livrărilor și a băuturilor folosesc alte 2 funcții ajutătoare ce transformă tipul de Beverage și de Delivery dintr-un struct într-un string pentru scrierea ușoară. Ele au ca parametri vectorul ce va fi scris în fișier și fișierul propriu zis. Din cauza că datele sunt într-un vector, se foloseste un for each loop pentru a itera prin tot vectorul și a scrie elementul în fișier.

```
void writeBeverages(const std::vector<Beverage> &beverages, const std::string
&filename) {
    std::ofstream file(filename);

    if (!file.is_open()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    for (auto &beverage : beverages) {
        file << beverageToString(beverage) << '\n';
    }

    file.close();
}
```

```
void writeDeliveries(const std::vector<Delivery> &deliveries, const std::string
&filename) {
    std::ofstream file(filename);

    if (!file.is_open()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    for (auto &delivery : deliveries) {
        file << deliveryToString(delivery) << '\n';
    }

    file.close();
}
```

Sunt doar 3 funcții ce ajută cu lucrarea cu strings. Primele 2 deja au fost menționate mai sus și ele convertesc tipul de băutura și livrare într-un string pentru scrierea și afișarea lor, iar a treia funcție permite crearea unei linii de strings repetate, folosită la crearea tabelelor.

strings.hpp

```
#pragma once
#include "../types/Beverages.hpp"
#include "../types/Delivery.hpp"
#include "string"

std::string createRepeatedString(const std::string lineStart, const std::string
repeatedString, const unsigned length,
                                 const std::string lineEnd);

std::string beverageToString(const Beverage &beverage);

std::string deliveryToString(const Delivery &delivery);
```

Funcțiile de convertire a datelor sunt foarte simple, folosind funcția standard de to_string din C++ pentru a converti fiecare element individual a băuturii/livrării și apoi le unește împreună într-o linie.

```
std::string beverageToString(const Beverage &beverage) {
    return std::to_string(beverage.id) + ' ' + std::to_string(beverage.type) + '
' + beverage.name + ' ' +
           beverage.color + ' ' + std::to_string(beverage.pricePerLiter);
};
```

```
std::string deliveryToString(const Delivery &delivery) {
    return std::to_string(delivery.id) + ' ' +
std::to_string(delivery.beverageId) + ' ' +
           std::to_string(delivery.quantityDelivered);
}
```

Funcția de create a unui string repetat, este folosit la crearea tabelelor în timpul afișării informațiilor despre băuturi și livrări. Ea are un caracter unic la început, unu la sfârșit și un caracter ce se repetă în mijloc pentru o lungime specificată.

```
std::string createRepeatedString(const std::string lineStart, const std::string
repeatedString, const unsigned length,
                                  const std::string lineEnd) {
    std::string line = lineStart;

    for (unsigned i = 0; i < length; i++) {
        line += repeatedString;
    }

    return line + lineEnd;
}
```

Funcțiile principale

În fiecare dosar din features (beverages și deliveries) există un fișier header cu același nume ce conține declarațiile a tuturor funcțiilor ce se află în acel dosar, iar apoi fiecare funcție își are propriul fișier .cpp în care este scrisă.

beverages.hpp

```
#pragma once
#include "string"

void displayBeverages(const std::string &filename);

void addNewBeverage(const std::string &filename);

void deleteBeverage(const std::string &filename);

void editBeverage(const std::string &filename);

void displaySpecificBeverage(const std::string &filename);

void displayBeverageValueExtremes(const std::string &beverageFile, const
std::string &deliveryFile);

void displayAveragePriceForQuantity(const std::string &beverageFile, const
std::string &deliveryFile);
```

deliveries.hpp

```
#pragma once
#include "string"

void displayDeliveries(const std::string &filename);

void addNewDelivery(const std::string &filename);

void deleteDelivery(const std::string &filename);

void editDelivery(const std::string &filename);

void summarizeDeliveries(const std::string &beverageFile, const std::string
&deliveryFile, const std::string &outFile);
```

Funcțiile de afișare a băuturilor și a livrărilor sunt aproape identice. Ele folosesc inițial funția ajutătoare de a citi conținuturile fișierelor de livrări și de băuturi, iterează prin vectorul întors pentru a afișa rezultatele. În același timp ele folosesc funcția ajutătoare pentru a crea stringuri repetate. Cu ajutorul bibiliotecii iomanip, programul crează tabele și le împarte în diferite secțiuni pentru afișare mai frumoasă.

```
void displayDeliveries(const std::string &filename) {
    std::vector<Delivery> deliveriesList = getAllDeliveries(filename);

    if (deliveriesList.empty()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    // Column widths
    const unsigned wId = 6;
    const unsigned wBeverageId = 12;
    const unsigned wQuantity = 18;

    // Top of the table
    std::cout << createRepeatedString("╔", "═", wId, "╦") <<
createRepeatedString("═", "═", wBeverageId - 1, "╦")
              << createRepeatedString("═", "═", wQuantity - 1, "╗\n");

    // Line displaying all table columns
    std::cout << std::left << "║" << std::setw(wId) << "ID" << "║" <<
std::setw(wBeverageId) << "ID Bautura"
              << "║" << std::setw(wQuantity) << "Cantitate livrata" << "║\n";

    // Divider between table header and data
    std::cout << createRepeatedString("╠", "═", wId, "╬") <<
createRepeatedString("═", "═", wBeverageId - 1, "╬")
              << createRepeatedString("═", "═", wQuantity - 1, "╣\n");

    // Display table data, information about all deliveries
    for (const auto &delivery : deliveriesList) {
        std::cout << std::left << "║" << std::setw(wId) << delivery.id << "║" <<
std::setw(wBeverageId)
                  << delivery.beverageId << "║" << std::setw(wQuantity) <<
delivery.quantityDelivered << "║\n";
    }

    // End of table
    std::cout << createRepeatedString("╚", "═", wId, "╩") <<
createRepeatedString("═", "═", wBeverageId - 1, "╩")
              << createRepeatedString("═", "═", wQuantity - 1, "╝\n");
};
```

```
void displayBeverages(const std::string &filename) {
    std::vector<Beverage> beverageList = getAllBeverages(filename);

    if (beverageList.empty()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    // Column widths
    const int wId = 6;
    const int wName = 25;
    const int wType = 17;
    const int wColor = 17;
```

```
    const int wPrice = 6;

    // Top of the table
    std::cout << createRepeatedString("╔", "═", wId, "╦") <<
createRepeatedString("═", "═", wName - 1, "╦")
              << createRepeatedString("═", "═", wType - 1, "╦") <<
createRepeatedString("═", "═", wColor - 1, "╦")
              << createRepeatedString("═", "═", wPrice - 1, "╗\n");

    // Line displaying all table columns
    std::cout << std::left << "║" << std::setw(wId) << "ID" << "║" <<
std::setw(wName) << "Nume" << "║"
              << std::setw(wType) << "Tip" << "║" << std::setw(wColor) <<
"Culoare" << "║" << std::setw(wPrice)
              << "Pret/L" << "║\n";

    // Divider between table header and data
    std::cout << createRepeatedString("╠", "═", wId, "╬") <<
createRepeatedString("═", "═", wName - 1, "╬")
              << createRepeatedString("═", "═", wType - 1, "╬") <<
createRepeatedString("═", "═", wColor - 1, "╬")
              << createRepeatedString("═", "═", wPrice - 1, "╣\n");

    // Display table data, information about all beverages
    for (const auto &bev : beverageList) {
        std::cout << std::left << "║" << std::setw(wId) << bev.id << "║" <<
std::setw(wName) << bev.name << "║"
                  << std::setw(wType) << getBeverageTypeName(bev.type) << "║" <<
std::setw(wColor) << bev.color << "║"
                  << std::setw(wPrice) << bev.pricePerLiter << "║\n";
    }

    // End of table
    std::cout << createRepeatedString("╚", "═", wId, "╩") <<
createRepeatedString("═", "═", wName - 1, "╩")
              << createRepeatedString("═", "═", wType - 1, "╩") <<
createRepeatedString("═", "═", wColor - 1, "╩")
              << createRepeatedString("═", "═", wPrice - 1, "╝\n");
};
```

Funcțiile de adăugare a băuturilor/livrărilor sunt foarte simple. Ele doar citesc informația de la utilizator, o convertesc într-un obiect cu tipul de Beverage, îl transformă în string folosind funcțiile ajutătoare și apoi introduc acest string la capătul fișierului de băuturi/livrări.

```
void addNewBeverage(const std::string &filename) {
    Beverage newBeverage{};
    int rawBeverageType;

    std::cout << "Introduceti ID bautura: ";
    std::cin >> newBeverage.id;

    std::cout << "Introduceti tip bautura (0=Apa
dulce,1=Minerala,2=Suc,3=Sirop): ";
    std::cin >> rawBeverageType;
    newBeverage.type = static_cast<BeverageType>(rawBeverageType);

    std::cout << "Introduceti numele bauturii: ";
    std::cin >> newBeverage.name;

    std::cout << "Introduceti culoarea bauturii: ";
    std::cin >> newBeverage.color;
```

```
    std::cout << "Introduceti pretul pe litru: ";
    std::cin >> newBeverage.pricePerLiter;

    std::string beverage = beverageToString(newBeverage);
    std::ofstream file(filename, std::ios::app);

    if (!file.is_open()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    file << beverage << '\n';
    file.close();
};
```

```
void addNewDelivery(const std::string &filename) {

    Delivery newDelivery{};

    std::cout << "Introduceti ID Livrare: ";
    std::cin >> newDelivery.id;

    std::cout << "Introduceti ID bautura livrata: ";
    std::cin >> newDelivery.beverageId;

    std::cout << "Introduceti cantitatea livrata: ";
    std::cin >> newDelivery.quantityDelivered;

    std::string delivery = deliveryToString(newDelivery);
    std::ofstream file(filename, std::ios::app);

    if (!file.is_open()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    file << delivery << '\n';
    file.close();
};
```

Funcțiile de ștergere a băuturilor/livrărilor sunt un pic mai complicate. Ele inițial citesc tot fișierul de băuturi/livrări folosind funcțiile ajutătoare, apoi folosind funcția standard de find_if, poate găsi în toate elementele vectorului elementul ce are condițiile pentru ștergere. Odată găsit, el este șters din vector, iar apoi fișierul este rescris cu noul vector ce nu mai conține elementul șters.

```
void deleteBeverage(const std::string &filename) {
    std::string name;

    std::cout << "Introdu numele bauturii pe care vrei sa o stergi: ";
    std::cin >> name;

    std::vector<Beverage> beverageList = getAllBeverages(filename);

    if (beverageList.empty()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    auto it = std::find_if(beverageList.begin(), beverageList.end(), [&](const
Beverage &b) { return b.name == name; });
```

```
    if (it == beverageList.end()) {
        std::cout << "Nu exista nici o bautura cu numele \"" << name << "\"\n";
        return;
    }

    beverageList.erase(it);
    writeBeverages(beverageList, filename);

    std::cout << "Sters bautura cu numele \"" << name << "\"\n";
}
```

```
void deleteDelivery(const std::string &filename) {
    unsigned id;

    std::cout << "Introdu ID-ul livrarii pe care vrei sa o stergi: ";
    std::cin >> id;

    std::vector<Delivery> deliveriesList = getAllDeliveries(filename);

    if (deliveriesList.empty()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    auto it = std::find_if(deliveriesList.begin(), deliveriesList.end(),
[&](const Delivery &d) { return d.id == id; });

    if (it == deliveriesList.end()) {
        std::cout << "Nu exista nici o livrare cu ID-ul \"" << id << "\"\n ";
        return;
    }

    deliveriesList.erase(it);
    writeDeliveries(deliveriesList, filename);

    std::cout << "Sters livrarea cu ID-ul \"" << id << "\"\n";
}
```

Editarea băuturilor/livrărilor este similară cu ștergerea, doar că înloc să șteargă elementul dat, programul iterează prin toate elementele vectorului până când găsește elementul ce are condiția de editare, îl editează în baza datelor introduse de utilizator și apoi rescrie tot vectorul în fișierul respectiv.

```
void editDelivery(const std::string &filename) {
    unsigned id;

    std::cout << "Introdu ID-ul livrarii pe care vrei sa o editezi: ";
    std::cin >> id;

    std::vector<Delivery> deliveries = getAllDeliveries(filename);

    for (auto &delivery : deliveries) {
        if (delivery.id != id) continue;

        std::cout << "Livrarea originala: " + deliveryToString(delivery) + '\n';
        std::cout << '\n';

        std::cout << "Modifica ID-ul livrarii: ";
        std::cin >> delivery.id;
```

```
        std::cout << "Modifica ID-ul bauturilor livrate: ";
        std::cin >> delivery.beverageId;

        std::cout << "Modifica cantitatea livrata: ";
        std::cin >> delivery.quantityDelivered;

        writeDeliveries(deliveries, filename);
        return;
    }

    std::cout << "Nu a fost gasita nici o livrare cu ID-ul '" << id << "'.\n";
}
```

```
void editBeverage(const std::string &filename) {
    unsigned id;

    std::cout << "Introdu ID-ul bauturii pe care vrei sa o editezi: ";
    std::cin >> id;

    std::vector<Beverage> beverages = getAllBeverages(filename);

    for (auto &beverage : beverages) {
        if (beverage.id != id) continue;

        std::cout << "Bautura originala: " + beverageToString(beverage) + '\n';
        std::cout << '\n';

        std::cout << "Modifica ID-ul bauturii: ";
        std::cin >> beverage.id;

        int rawBeverageType;
        std::cout << "Modifica tipul bauturii(0=Apa
dulce,1=Minerala,2=Suc,3=Sirop): ";
        std::cin >> rawBeverageType;
        beverage.type = static_cast<BeverageType>(rawBeverageType);

        std::cout << "Modifica numele bauturii: ";
        std::cin >> beverage.name;

        std::cout << "Modifica culoarea bauturii: ";
        std::cin >> beverage.color;

        std::cout << "Modifica pretul pe litru al bauturii: ";
        std::cin >> beverage.pricePerLiter;

        writeBeverages(beverages, filename);
        return;
    }

    std::cout << "Nu a fost gasita nici o bautura cu ID-ul '" << id << "'.\n";
}
```

Livrările au o opțiune de creare a unu rezumat ce conține ID-ul băuturii, cantitatea total livrată și valoarea totală de bani a acestor livrări. Acest lucru poate fi ușor făcut prin crearea unui nou tip ce are rezumatul doar unei livrări, crearea unui vector de rezumate și folosind 2 for each bucle, putem itera mai întâi prin băuturi și apoi prin livrări pentru a obține informațiile necesare.

```
struct Summary {
    std::string name;
    double quantityDelivered;
    double value;
};

void summarizeDeliveries(const std::string &beverageFile, const std::string
&deliveryFile, const std::string &outFile) {
    std::vector<Beverage> beverages = getAllBeverages(beverageFile);
    std::vector<Delivery> deliveries = getAllDeliveries(deliveryFile);

    if (beverages.empty()) {
        std::cerr << "Error: cannot open file \"" << beverageFile << "\"\n";
        return;
    }

    if (deliveries.empty()) {
        std::cerr << "Error: cannot open file \"" << deliveryFile << "\"\n";
        return;
    }

    std::vector<Summary> summaries;
    summaries.reserve(beverages.size());

    for (const auto &beverage : beverages) {
        Summary newSummary{name : beverage.name, quantityDelivered : 0, value :
0};

        for (const auto &delivery : deliveries) {
            if (delivery.beverageId != beverage.id) continue;

            newSummary.quantityDelivered += delivery.quantityDelivered;
        }

        newSummary.value = newSummary.quantityDelivered *
beverage.pricePerLiter;

        summaries.push_back(newSummary);
    }

    std::ofstream summaryFile(outFile);

    if (!summaryFile.is_open()) {
        std::cerr << "Error: cannot open file \"" << outFile << "\"\n";
        return;
    }

    for (const auto &summary : summaries) {
        summaryFile << summary.name << ' ' <<
std::to_string(summary.quantityDelivered) << ' '
                    << std::to_string(summary.value) << '\n';
    }

    std::cout << "Creat rezumatul livrarilor in fisierul \"" << outFile <<
"\"\n";
    summaryFile.close();
}
```

Băuturile au o opțiune de a afișa doar băuturile de un anumit tip și în același timp să le afișeze în ordine alfabetică. Acest lucru este posibil folosind funcția ajutătoare de a obține băturile doar de un anumit tip și apoi sortând-ule folosind insertion sort.

```
// Sort beverage vector using insertion sort alphabetically
void sortByName(std::vector<Beverage> &vector) {
    for (unsigned i = 1; i < vector.size(); ++i) {
        Beverage key = vector[i];

        unsigned j = i;
        while (j > 0 && vector[j - 1].name > key.name) {
            vector[j] = vector[j - 1];
            --j;
        }

        vector[j] = key;
    }
}

void displaySpecificBeverage(const std::string &filename) {
    int rawBeverageType;
    std::cout << "Introdu tipul de bauturi ce sa fie afisate (0=Apa
dulce,1=Minerala,2=Suc,3=Sirop): ";
    std::cin >> rawBeverageType;

    BeverageType beverageType = static_cast<BeverageType>(rawBeverageType);
    std::vector<Beverage> beverages = getSpecificBeverage(filename,
beverageType);

    if (beverages.empty()) {
        std::cerr << "Error: cannot open file \"" << filename << "\"\n";
        return;
    }

    sortByName(beverages);

    const int wId = 6;
    const int wName = 25;
    const int wType = 17;
    const int wColor = 17;
    const int wPrice = 6;

    // Top of the table
    std::cout << createRepeatedString("╔", "=", wId, "╦") <<
createRepeatedString("=", "=", wName - 1, "╦")
              << createRepeatedString("=", "=", wType - 1, "╦") <<
createRepeatedString("=", "=", wColor - 1, "╦")
              << createRepeatedString("=", "=", wPrice - 1, "╗\n");

    // Line displaying all table columns
    std::cout << std::left << "║" << std::setw(wId) << "ID" << "║" <<
std::setw(wName) << "Nume" << "║"
              << std::setw(wType) << "Tip" << "║" << std::setw(wColor) <<
"Culoare" << "║" << std::setw(wPrice)
              << "Pret/L" << "║\n";

    // Divider between table header and data
    std::cout << createRepeatedString("╠", "=", wId, "╬") <<
createRepeatedString("=", "=", wName - 1, "╬")
              << createRepeatedString("=", "=", wType - 1, "╬") <<
createRepeatedString("=", "=", wColor - 1, "╬")
              << createRepeatedString("=", "=", wPrice - 1, "╣\n");
```

```
    // Display table data, information about all beverages
    for (const auto &bev : beverages) {
        std::cout << std::left << "╟" << std::setw(wId) << bev.id << "╟" <<
std::setw(wName) << bev.name << "╟"
                  << std::setw(wType) << getBeverageTypeName(bev.type) << "╟" <<
std::setw(wColor) << bev.color << "╟"
                  << std::setw(wPrice) << bev.pricePerLiter << "╟\n";
    }

    // End of table
    std::cout << createRepeatedString("╚", "═", wId, "╩") <<
createRepeatedString("═", "═", wName - 1, "╩")
              << createRepeatedString("═", "═", wType - 1, "╩") <<
createRepeatedString("═", "═", wColor - 1, "╩")
              << createRepeatedString("═", "═", wPrice - 1, "╝\n");
}
```

O altă opțiune a băuturilor este abilitatea de a afișa prețul mediu pentru o cantiate minimă livrată. Acest lucru este efectuat folosind inițial 2 for each bucle pentru a determina cantitatea totală a tuturor băuturilor. Apoi folosind un alt for each loop se determină toate băuturile ce au livrări în cantitate mai marea ca cea minimă și apoi media este calculată.

```
void displayAveragePriceForQuantity(const std::string &beverageFile, const
std::string &deliveryFile) {
    std::vector<Beverage> beverages = getAllBeverages(beverageFile);
    std::vector<Delivery> deliveries = getAllDeliveries(deliveryFile);

    if (beverages.empty()) {
        std::cerr << "Error: cannot open file \"" << beverageFile << "\"\n";
        return;
    }

    if (deliveries.empty()) {
        std::cerr << "Error: cannot open file \"" << deliveryFile << "\"\n";
        return;
    }

    double minQuantity;
    std::cout << "Introdu cantitatea minimă livrată (litri): ";
    std::cin >> minQuantity;

    // Holds the total quantity of all beverages. beverages[i] corresponds to
totalQuantity[i]
    std::vector<double> totalQuantity;
    totalQuantity.reserve(beverages.size());

    for (const auto &beverage : beverages) {
        double quantity = 0.0;

        for (const auto &delivery : deliveries) {
            if (beverage.id != delivery.beverageId) continue;

            quantity += delivery.quantityDelivered;
        }

        totalQuantity.push_back(quantity);
    }

    double sumPrices = 0.0;
    unsigned count = 0;
```

```
    // Pointer to the totalQuantity vector
    auto quantityPointer = totalQuantity.begin();

    for (const auto &bev : beverages) {
        // After every iteration, increase the pointer so that it now points to
the next element in the vector
        double quantity = *quantityPointer++;

        if (quantity <= minQuantity) continue;

        sumPrices += bev.pricePerLiter;
        ++count;
    }

    if (count == 0) {
        std::cout << "Nicio băutură nu are livrări ≥ " << minQuantity << "
litri.\n";
        return;
    }

    double averagePrice = sumPrices / count;
    std::cout << "Media pretului pe litru a bauturilor cu livrari ≥ " <<
minQuantity << "L este: " << averagePrice
              << " lei/L\n";
}
```

Ultima opțiune a băuturilor este cea de a afișa extremele valorilor (băuturea cu cea mai mare și cea mai mică valoare). Acest lucru este făcut folosind 2 for each bucle pentru a determina valoare fiecărei băutură, apoi se verifică cu cea mai mică și cea mai mare livrare până atunci. În final ele sunt afișate într-un tabel.

```
void displayBeverageValueExtremes(const std::string &beverageFile, const
std::string &deliveryFile) {
    std::vector<Beverage> beverages = getAllBeverages(beverageFile);
    std::vector<Delivery> deliveries = getAllDeliveries(deliveryFile);

    if (beverages.empty()) {
        std::cerr << "Error: cannot open file \"" << beverageFile << "\"\n";
        return;
    }

    if (deliveries.empty()) {
        std::cerr << "Error: cannot open file \"" << deliveryFile << "\"\n";
        return;
    }

    Beverage highestValueBeverage = beverages[0];
    double higestValue = 0;

    Beverage lowestValueBeverage = beverages[0];
    double lowestValue = 10000;

    for (const auto &beverage : beverages) {
        double quantityDelivered = 0;

        for (const auto &delivery : deliveries) {
            if (delivery.beverageId != beverage.id) continue;

            quantityDelivered += delivery.quantityDelivered;
        }
```

```
        double value = quantityDelivered * beverage.pricePerLiter;

        if (value > higestValue) {
            higestValue = value;
            highestValueBeverage = beverage;
        }

        if (value < lowestValue) {
            lowestValue = value;
            lowestValueBeverage = beverage;
        }
    }

    const int wId = 6;
    const int wName = 25;
    const int wType = 17;
    const int wColor = 17;
    const int wPrice = 6;
    const int wValue = 10;

    // Top of the table
    std::cout << createRepeatedString("╔", "═", wId, "╦") <<
createRepeatedString("═", "═", wName - 1, "╦")
              << createRepeatedString("═", "═", wType - 1, "╦") <<
createRepeatedString("═", "═", wColor - 1, "╦")
              << createRepeatedString("═", "═", wPrice - 1, "╦") <<
createRepeatedString("═", "═", wValue - 1, "╗\n");

    // Line displaying all table columns
    std::cout << std::left << "║" << std::setw(wId) << "ID" << "║" <<
std::setw(wName) << "Nume" << "║"
              << std::setw(wType) << "Tip" << "║" << std::setw(wColor) <<
"Culoare" << "║" << std::setw(wPrice)
              << "Pret/L" << "║" << std::setw(wValue) << "Valoare" << "║\n";

    // Divider between table header and data
    std::cout << createRepeatedString("╠", "═", wId, "╬") <<
createRepeatedString("═", "═", wName - 1, "╬")
              << createRepeatedString("═", "═", wType - 1, "╬") <<
createRepeatedString("═", "═", wColor - 1, "╬")
              << createRepeatedString("═", "═", wPrice - 1, "╬") <<
createRepeatedString("═", "═", wValue - 1, "╣\n");

    std::cout << std::left << "║" << std::setw(wId) << highestValueBeverage.id
<< "║" << std::setw(wName)
              << highestValueBeverage.name << "║" << std::setw(wType) <<
getBeverageTypeName(highestValueBeverage.type)
              << "║" << std::setw(wColor) << highestValueBeverage.color << "║"
<< std::setw(wPrice)
              << highestValueBeverage.pricePerLiter << "║" << std::setw(wValue)
<< higestValue << "║\n";

    std::cout << std::left << "║" << std::setw(wId) << lowestValueBeverage.id <<
"║" << std::setw(wName)
              << lowestValueBeverage.name << "║" << std::setw(wType) <<
getBeverageTypeName(lowestValueBeverage.type)
              << "║" << std::setw(wColor) << lowestValueBeverage.color << "║" <<
std::setw(wPrice)
              << lowestValueBeverage.pricePerLiter << "║" << std::setw(wValue)
<< lowestValue << "║\n";

    // End of table
```

```
    std::cout << createRepeatedString("╚", "═", wId, "╩") <<
createRepeatedString("═", "═", wName - 1, "╩")
              << createRepeatedString("═", "═", wType - 1, "╩") <<
createRepeatedString("═", "═", wColor - 1, "╩")
              << createRepeatedString("═", "═", wPrice - 1, "╩") <<
createRepeatedString("═", "═", wValue - 1, "╝\n");
}
```

Baut				
id_baut	Tip	Denumire	Culoare	Pret/Litru
1	0	CocaCola	MaroInchis	MDL 4.20
2	0	Fanta	Portocaliu	MDL 3.80
3	0	Sprite	Transparent	MDL 3.50
4	0	Pepsi	MaroInchis	MDL 4.00
5	1	Dorna	Transparent	MDL 1.15
6	1	Jana	Transparent	MDL 1.10
7	1	OM	Transparent	MDL 1.05
8	1	GuraCainarului	Transparent	MDL 1.20
9	2	Naturalis	Portocaliu	MDL 5.00
10	2	RichMere	Rosu	MDL 4.80
11	2	SunnyMacedonia	Galben	MDL 5.20
12	2	MagicOrange	Portocaliu	MDL 5.10
13	3	Grenadine	Rosu	MDL 2.60
14	3	Menta	Verde	MDL 2.80
15	3	Zmeura	Rosu	MDL 3.00

Figura 14

id_livrare	id_baut	Cantit	Suma
1	1	12.50	€ 2.69
2	4	8.75	€ 1.79
3	7	15.00	€ 0.81
4	2	5.25	€ 1.02
5	10	9.00	€ 2.21
6	5	11.30	€ 0.66
7	8	7.80	€ 0.48
8	3	14.20	€ 2.54
9	6	6.50	€ 0.37
10	9	10.00	€ 2.56
11	12	4.40	€ 1.15
12	15	13.70	€ 2.10
13	11	9.25	€ 2.46
14	14	8.10	€ 1.16
15	13	16.00	€ 2.13
Total		**151.75**	**€ 24.12**

Figura 15

În figurile 14 și 15 sunt arătate tabele principale ce conțin datele direct importate din Baut.txt și Livr.txt.

Tipul, denumirea și culoarea sunt selectate dintr-o listă derulantă folosind data validation.

În figura 15, ultima coloana de sumă, automat calculează suma livrării în euro în baza prețului pe litru din primul tabel aflat în Figura 14.

Bifa	Status	Denumire		Cantitatea total livrata	Suma total achitata
☐	FALSE	CocaCola		0	MDL 0.00
☐	FALSE	Fanta			
☐	FALSE	Sprite			
☐	FALSE	Pepsi			
☐	FALSE	Dorna			
☐	FALSE	Jana			
☐	FALSE	OM			
☐	FALSE	GuraCainarului			
☐	FALSE	Naturalis			
☐	FALSE	RichMere			
☐	FALSE	SunnyMacedonia			
☐	FALSE	MagicOrange			
☐	FALSE	Grenadine			
☐	FALSE	Menta			
☐	FALSE	Zmeura			

Figura 16

În figurile 16 și 17 sunt arătate cantiatea totală livrată de băutură și suma totală achitată pentru băuturile selectate de utlizator din tabel.

El folosește tabelul 2 cu livrările pentru a obține cantitatea totală livrată și suma de bani, iar apoi din euro le convertește înapoi în MDL.

Bifa	Status	Denumire		Cantitatea total livrata	Suma total achitata
☐	FALSE	CocaCola		26.55	MDL 55.15
☐	FALSE	Fanta			
☐	FALSE	Sprite			
☑	TRUE	Pepsi			
☑	TRUE	Dorna			
☑	TRUE	Jana			
☐	FALSE	OM			
☐	FALSE	GuraCainarului			
☐	FALSE	Naturalis			
☐	FALSE	RichMere			
☐	FALSE	SunnyMacedonia			
☐	FALSE	MagicOrange			
☐	FALSE	Grenadine			
☐	FALSE	Menta			
☐	FALSE	Zmeura			

Figura 17

În figura 18 se poate vedea diagrama cu toate băuturile și suma lor achitată în livrări.

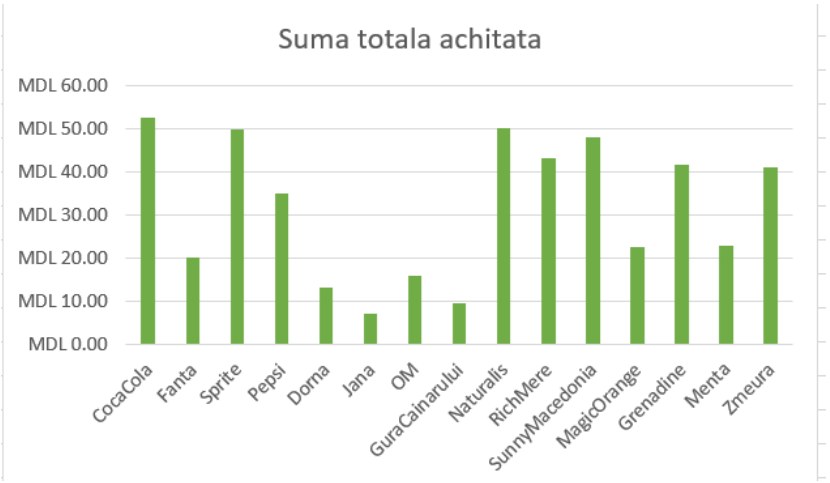

Figura 18

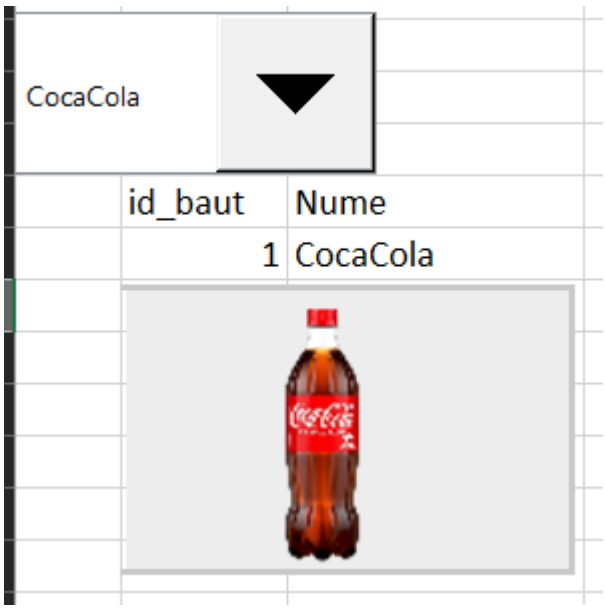

Figura 19

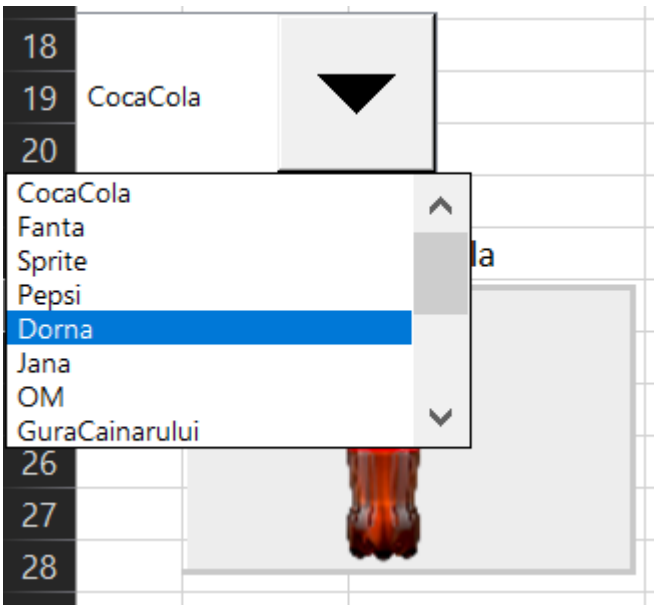

Figura 20

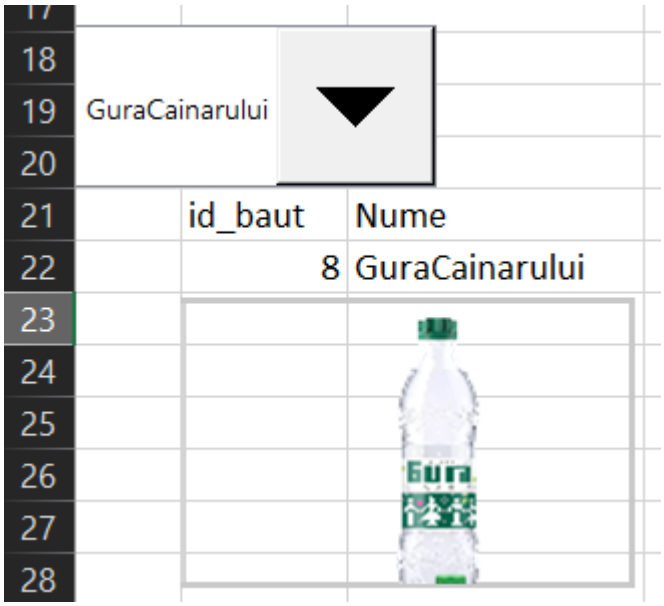

Figura 21

În figurile 19, 20 și 21 este arătat funcționalitatea combo box. Utilizatorul poate alege o băutură și îi va fi afișat numele ei, ID-ul băuturii și poza ei.

Concluzie

În concluzie, produsul program „Băuturi” a demonstrat eficiența combinării limbajului C++ cu Microsoft Excel pentru gestionarea stocurilor și livrărilor de băuturi răcoritoare. În C++ am folosit biblioteca *fstream* pentru operațiuni de citire și scriere directă în fișiere text, în special containerele de tip *vector*, pentru a stoca dinamic în memorie înregistrările despre produse și livrări. Utilizarea vectorilor a permis implementarea ușoară a funcțiilor de adăugare, modificare, ștergere și sortare, precum și generarea rapoartelor sumare, menținând totodată o arhitectură modulară, clară și ușor de extins.

Partea Excel a adus un plus de interactivitate și vizualizare: importul automat al datelor, validarea cu liste derulante și calculul imediat al indicatorilor (stocuri disponibile, valoarea livrărilor) au fost completate de grafice care ilustrează evoluția volumelor și a câștigurilor.

Gradul de complexitate al proiectului a fost moderat, concentrându-se pe gestionarea eficientă a fișierelor text și pe integrarea rapidă a datelor în Excel. Rezultatul este o aplicație funcțională, prietenoasă cu utilizatorul și solidă din punct de vedere tehnic.

Ca direcții de dezvoltare ulterioară, se pot explora crearea unei interfețe grafice pentru C++, migrarea către o bază de date relațională pentru o stocare mai structurată, automatizarea schimbului de date între C++ și Excel prin scripturi sau API-uri externe, implementarea de filtre și căutări avansate în cadrul aplicației și optimizarea operațiilor de ștergere pentru consum minim de resurse. Aceste îmbunătățiri vor transforma soluția actuală într-un instrument și mai robust și versatil.

Webografie

https://www.microsoft.com/en-us/microsoft-365/

https://code.visualstudio.com/

https://gcc.gnu.org/

https://www.w3schools.com/cpp/cpp_vectors.asp

https://cplusplus.com/reference/algorithm/find_if/

https://cplusplus.com/reference/string/string/

Anexe

Anexa 1
https://github.com/maxpricop/max_programare_ceiti/tree/main/programareProcedurala/practica

Anexa 2
https://drive.google.com/drive/folders/1wMhsxsQqJWK7RTHvYtuuyQPZLPfl6t0l?usp=sharing