Ministerul Educației, Culturii și Cercetării al Republicii Moldova

IP Centrul de Excelență în Informatică și Tehnologii Informaționale

**A blue letter on a black background

AI-generated content may be incorrect.**

**RAPORTUL**

**STAGIULUI DE PRACTICĂ**

Elevul:  **Orlețchi Bogdan**

Grupa: **P-2412**

Specialitatea: **Programarea și testarea produselor program**

Baza de practică: **IP.CEITI**

Conducătorul stagiului de practică

de la unitatea economică

**Gîrlea Elena**

Director adjunct pentru instruirea practică și producere

**Vitalie Zavadschi**

**Cuprins**

[Introducere 3](#_Toc198838476)

[Schema bloc generală a algoritmului elaborat 4](#_Toc198838477)

[Cerințele lucrului individual 5](#_Toc198838478)

[Codul Programului 7](#_Toc198838479)

[Datele de intrare ( Emisiuni.txt; Canal.txt ) 24](#_Toc198838480)

[Date de ieșire (BroadcastSummary.txt) 26](#_Toc198838481)

[Capturi de ecran cu reprezentare lucrului programului 27](#_Toc198838482)

[Concluzie 31](#_Toc198838483)

[Bibliografie 32](#_Toc198838484)

# Introducere

În acest raport voi prezenta, pas cu pas, activitățile desfășurate pe parcursul stagiului de practică din primul an de studiu, specializarea programarea și testarea produselor program.

Obiectivul principal al acestei experiențe a constat în consolidarea și evaluarea competențelor dobândite la cursuri, precum și în înțelegerea modului în care teoriile studiate se aplică concret în programarea și testarea produselor program. Prin implicarea activă în proiecte practice, am avut ocazia să îmi extind orizonturile profesionale și să mă familiarizez cu provocările specifice domeniului.

Această introducere conturează scopul raportului și oferă o privire de ansamblu asupra obiectivelor și realizărilor mele din perioada de practică.

# Schema bloc generală a algoritmului elaborat

A diagram of a flowchart

AI-generated content may be incorrect.

1. Schema bloc a algoritmului

# Cerințele lucrului individual

Informația despre emisiuni (nume emisiune, categorie, oră de început, durată, ziua săptămâni, cod canal) se păstrează în fișierul text Program.txt. Fiecare linie a fișierului conține informația despre o singura emisiune și invers.

Informația despre canale TV (cod canal, denumirea canalului, țara de origine) se păstrează în fișierul text Channel.txt. Fiecare linie a fișierului conține informația despre un canal TV și invers.

C++ Să se elaboreze un program (modular) care, folosind meniuri și subprograme, să realizeze, la solicitarea utilizatorului, următoarele prescripții

1. Afișarea pe ecran a informației despre toate emisiunile.
2. Afișarea pe ecran a informației despre toate canalele TV.
3. Adăugarea unei noi emisiuni.
4. Adăugarea unui canal TV nou.
5. Ștergerea unei emisiuni, al cărei nume este introdus de la tastatură.
6. Ștergerea unui canal TV, al cărui nume este introdus de la tastatură.
7. Editarea informației despre o emisiune, al cărei nume este introdus de la tastatură.
8. Editarea informației despre un canal TV, al cărui nume este introdus de la tastatură.
9. Crearea fișierului text BroadcastSummary.txt, care va conține denumirea fiecărui canal și numărul total de emisiuni difuzate.
10. Afișarea tuturor emisiunilor dintr-o i specificată, în ordine cronologică.
11. Afișarea emisiunii cu durata maximă și a celei cu durata minimă.
12. Afișarea duratei medii a emisiunilor pentru o anumită categorie.

MS Excel Realizați următoarele sarcini în aplicația MS Excel:

1. Creează fișierul TVPrograms. Efectuează importul datelor din fișierele text Program.txt și Channel.txt în foi separate cu denumiri corespunzătoare.
2. Adaugă un titlu pentru fiecare tabel și denumește coloanele în mod corespunzător.
3. Formatează tabelele: setează chenarul, culoarea, dimensiunea și tipul fontului etc.
4. Valorile cod canal și categoria emisiuni din tabelul Program, respectiv tara din tabelul Channel trebuie să fie selectabile dintr-o listă derulantă.
5. in tabelul Program, adaugă o coloană suplimentară care să conțină fotografia emisiunii Pentru titlul tabelului Channel, adaugă un comentariu care să conțină logoul postului de televiziune.
6. În tabelul Channel, adaugă o coloană nouă cu numărul total de emisiuni difuzate de fiecare canal, precum și o linie nouă care să conțină totalul general al emisiunilor.
7. Calculează numărul de emisiuni și durata medie a acestora pentru fiecare zi a săptămâni. Salvează calculele într-o foaie numită Statistici.
8. 8 Reprezintă grafic numărul de emisiuni difuzate in fiecare zi a săptămânii.
9. Afișează informațiile despre toate emisiunile difuzate de canalul selectat dintr-o listă derulantă (ComboBox).
10. Afișează fotografia emisiunii al cărei nume este selectat dintr-o listă derulantă.

MS Word, MS Power Point

Creați un raport la practică în MS Word conform cerințelor predefinite. Creați o prezentare in MS POWER POINT, care va conține o relatare succintă des pre proiectele realizate.

# Codul Programului

#include "tvmodule.h"  
#include <iostream>  
#include <fstream>  
#include <algorithm>  
#include <map>  
#include <iomanip>  
#include <filesystem>  
  
using namespace std;  
  
// Define global containers  
vector<show> programs;  
vector<channel> channels;  
  
// Function to clear the screen (cross-platform)  
void clearScreen() {  
#ifdef \_WIN32  
 system("cls");  
#else  
 system("clear");  
#endif  
}  
  
static string encode(const string& s) {  
 string r;  
 for (char c : s) r += (c == ' ' ? '\_' : c);  
 return r;  
}  
  
static string decode(const string& s) {  
 string r;  
 for (char c : s) r += (c == '\_' ? ' ' : c);  
 return r;  
}  
  
bool fileExists(const string& fileName) {  
 ifstream file(fileName);  
 return file.good();  
}  
  
void createFileIfNotExists(const string& fileName) {  
 if (!fileExists(fileName)) {  
 ofstream file(fileName);  
 file.close();  
 cout << "Created new file: " << fileName << endl;  
 }  
}  
  
string generateNextChannelId() {  
 int maxId = 0;  
   
 // Find the highest existing ID  
 for (const auto& c : channels) {  
 try {  
 int currentId = stoi(c.code);  
 maxId = max(maxId, currentId);  
 } catch (const invalid\_argument&) {  
 // Skip non-numeric IDs  
 } catch (const out\_of\_range&) {  
 // Skip IDs that are too large  
 }  
 }  
   
 // Return next ID as string  
 return to\_string(maxId + 1);  
}  
  
void allShows() {  
 if (programs.empty()) {  
 cout << "No shows available." << endl;  
 return;  
 }  
  
 // First pass: determine needed column widths based on content  
 int nameWidth = 4; // minimum width for "Name"  
 int categoryWidth = 8; // minimum width for "Category"  
 int timeWidth = 10; // minimum width for "Start Time"  
 int durationWidth = 8; // minimum width for "Duration"  
 int dayWidth = 3; // minimum width for "Day"  
 int channelWidth = 12; // minimum width for "Channel Code"  
  
 // Determine maximum content width for each column  
 for (const auto& s : programs) {  
 nameWidth = max(nameWidth, static\_cast<int>(decode(s.name).length()));  
 categoryWidth = max(categoryWidth, static\_cast<int>(decode(s.category).length()));  
 dayWidth = max(dayWidth, static\_cast<int>(decode(s.dayOfWeek).length()));  
 channelWidth = max(channelWidth, static\_cast<int>(s.channelCode.length()));  
  
 // Calculate duration string length and consider it for column width  
 string durationStr = to\_string(s.duration) + " min";  
 durationWidth = max(durationWidth, static\_cast<int>(durationStr.length()));  
  
 // Calculate time string length  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 timeWidth = max(timeWidth, static\_cast<int>(startTime.length()));  
 }  
  
 // Add padding (1 space on each side)  
 nameWidth += 2;  
 categoryWidth += 2;  
 timeWidth += 2;  
 durationWidth += 2;  
 dayWidth += 2;  
 channelWidth += 2;  
  
 // Print header  
 int totalWidth = nameWidth + categoryWidth + timeWidth + durationWidth + dayWidth + channelWidth + 7; // 7 for the separators  
  
 cout << endl << string(totalWidth, '-') << endl;  
 cout << "|" << setw(nameWidth) << left << " Name"  
 << "|" << setw(categoryWidth) << " Category"  
 << "|" << setw(timeWidth) << " Start Time"  
 << "|" << setw(durationWidth) << " Duration"  
 << "|" << setw(dayWidth) << " Day"  
 << "|" << setw(channelWidth) << " Channel Code" << "|" << endl;  
 cout << string(totalWidth, '-') << endl;  
  
 // Print data rows  
 for (const auto& s : programs) {  
 string name = decode(s.name);  
 string category = decode(s.category);  
 string day = decode(s.dayOfWeek);  
 string channelCode = s.channelCode;  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 string durationStr = to\_string(s.duration) + " min";  
  
 cout << "|" << setw(nameWidth) << " " + name  
 << "|" << setw(categoryWidth) << " " + category  
 << "|" << setw(timeWidth) << " " + startTime  
 << "|" << setw(durationWidth) << " " + durationStr  
 << "|" << setw(dayWidth) << " " + day  
 << "|" << setw(channelWidth) << " " + channelCode << "|" << endl;  
 }  
  
 cout << string(totalWidth, '-') << endl;  
 cout << programs.size() << " shows found." << endl;  
}  
  
void allChannels() {  
 if (channels.empty()) {  
 cout << "No channels available." << endl;  
 return;  
 }  
  
 // First pass: determine needed column widths based on content  
 int codeWidth = 4; // minimum width for "Code"  
 int nameWidth = 4; // minimum width for "Name"  
 int countryWidth = 17; // minimum width for "Country"  
  
 // Determine maximum content width for each column  
 for (const auto& c : channels) {  
 codeWidth = max(codeWidth, static\_cast<int>(c.code.length()) + 1);  
 nameWidth = max(nameWidth, static\_cast<int>(c.name.length()) + 1);  
 countryWidth = max(countryWidth, static\_cast<int>(c.originCountry.length()) + 1);  
 }  
  
 // Print header  
 int totalWidth = codeWidth + nameWidth + countryWidth + (3 \* 3) + 1;  
 cout << string(totalWidth, '-') << endl;  
 cout << "| " << left << setw(codeWidth) << "Code"  
 << " | " << setw(nameWidth) << "Name"  
 << " | " << setw(countryWidth) << "Country of Origin" << " |" << endl;  
 cout << string(totalWidth, '-') << endl;  
  
 // And update the data rows to match:  
 for (const auto& c : channels) {  
 cout << "| " << setw(codeWidth) << c.code  
 << " | " << setw(nameWidth) << decode(c.name)  
 << " | " << setw(countryWidth) << decode(c.originCountry) << " |" << endl;  
 }  
  
 cout << string(totalWidth, '-') << endl;  
 cout << channels.size() << " channels found." << endl;  
}  
  
void addShow(const string& name, const string& category, const string& startTime, int duration, const string& dayOfWeek, string channelCode) {  
 if (name.empty() || category.empty() || startTime.empty() || duration <= 0 || dayOfWeek.empty() || channelCode.empty()) {  
 cout << "Invalid input. Please provide valid show details." << endl;  
 return;  
 }  
  
 string encName = encode(name);  
 string encCategory = encode(category);  
 string encDay = encode(dayOfWeek);  
  
 if (ranges::any\_of(programs, [&encName](const show& s) { return s.name == encName; })) {  
 cout << "Show with this name already exists." << endl;  
 return;  
 }  
  
 bool channelExists = ranges::any\_of(channels, [&channelCode](const channel& c) { return c.code == channelCode; });  
 if (!channelExists) {  
 cout << "Error: Channel code does not exist. Please enter a valid channel code." << endl;  
 return;  
 }  
  
 int startHour = 0, startMinute = 0;  
 size\_t colonPos = startTime.find(':');  
 if (colonPos != string::npos) {  
 try {  
 startHour = stoi(startTime.substr(0, colonPos));  
 startMinute = stoi(startTime.substr(colonPos + 1));  
  
 // Validate time ranges  
 if (startHour < 0 || startHour > 23 || startMinute < 0 || startMinute > 59) {  
 cout << "Invalid time. Hours must be 0-23, minutes must be 0-59." << endl;  
 return; // or set to default values  
 }  
 } catch (const exception& e) {  
 cout << "Invalid time format: " << e.what() << endl;  
 return; // or set to default values  
 }  
 }  
  
 show s;  
 s.name = encName;  
 s.category = encCategory;  
 s.startHour = startHour;  
 s.startMinute = startMinute;  
 s.duration = duration;  
 s.dayOfWeek = encDay;  
 s.channelCode = channelCode;  
 programs.push\_back(s);  
  
 createFileIfNotExists("Program.txt");  
 ofstream o("Program.txt", ios::app);  
 o << s.name << ' ' << s.category << ' '  
 << (s.startHour < 10 ? "0" : "") << s.startHour << ':'  
 << (s.startMinute < 10 ? "0" : "") << s.startMinute << ' '  
 << s.duration << ' ' << s.dayOfWeek << ' ' << s.channelCode << endl;  
 o.close();  
  
 cout << "Show added successfully." << endl;  
}  
  
void addChannel(const string& name, const string& originCountry) {  
 if (name.empty() || originCountry.empty()) {  
 cout << "Invalid input. Please provide valid channel details." << endl;  
 return;  
 }  
  
 string encName = encode(name);  
 string encCountry = encode(originCountry);  
  
 if (ranges::any\_of(channels, [&encName](const channel& c) { return c.name == encName; })) {  
 cout << "Channel with this name already exists." << endl;  
 return;  
 }  
  
 string code = generateNextChannelId();  
   
 channel c;  
 c.code = code;  
 c.name = encName;  
 c.originCountry = encCountry;  
 channels.push\_back(c);  
  
 createFileIfNotExists("Channel.txt");  
 ofstream o("Channel.txt", ios::app);  
 o << c.code << ' ' << c.name << ' ' << c.originCountry << endl;  
 o.close();  
   
 cout << "Channel added successfully with ID: " << code << endl;  
}  
  
void deleteShow(const string& name) {  
 if (name.empty()) {  
 cout << "Invalid input. Please provide valid show details." << endl;  
 return;  
 }  
  
 string encName = encode(name);  
 auto it = ranges::remove\_if(programs, [&encName](const show& s) { return s.name == encName; }).begin();  
 if (it != programs.end()) {  
 programs.erase(it, programs.end());  
 cout << "Show deleted successfully." << endl;  
 } else {  
 cout << "Show not found." << endl;  
 }  
 if (!fileExists("Program.txt")) {  
 cout << "File not found. Cannot update." << endl;  
 return;  
 }  
 ofstream o("Program.txt");  
 for (const auto& s : programs) {  
 string startTimeStr = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 o << s.name << " " << s.category << " " << startTimeStr << " " << s.duration << " " << s.dayOfWeek << " " << s.channelCode << endl;  
 }  
}  
  
void deleteChannel(const string& name) {  
 if (name.empty()) {  
 cout << "Invalid input. Please provide valid show details." << endl;  
 return;  
 }  
  
 string encName = encode(name);  
 auto it = ranges::remove\_if(channels, [&encName](const channel& c) { return c.name == encName; }).begin();  
 if (it != channels.end()) {  
 channels.erase(it, channels.end());  
 cout << "Channel deleted successfully." << endl;  
 } else {  
 cout << "Channel not found." << endl;  
 }  
 if (!fileExists("Channel.txt")) {  
 cout << "File not found. Cannot update." << endl;  
 return;  
 }  
 ofstream o("Channel.txt");  
 for (const auto& c : channels) {  
 o << c.code << " " << c.name << " " << c.originCountry << endl;  
 }  
}  
  
void editShow(const string& name, string newName, string newCategory, string newStartTime, int newDuration, string newDayOfWeek, string newChannelCode) {  
 if (name.empty()) {  
 cout << "Invalid input. Please provide valid show details." << endl;  
 return;  
 }  
 string encName = encode(name);  
  
 auto it = ranges::find\_if(programs, [&encName](const show& s) { return s.name == encName; });  
 if (it != programs.end()) {  
 cout << "Editing show: " << decode(it->name) << endl;  
  
 if (newName.empty() || newCategory.empty() || newStartTime.empty() ||  
 newDuration == 0 || newDayOfWeek.empty() || newChannelCode.empty()) {  
 cout << "Enter new details (leave blank to keep current value):" << endl;  
 }  
  
 if (newName.empty()) {  
 cout << "Name: ";  
 getline(cin, newName);  
 newName = encode(newName);  
 if (!newName.empty() && ranges::any\_of(programs, [&](const show& s) {  
 return s.name == newName && s.name != name; })) {  
 cout << "Show with this name already exists." << endl;  
 return;  
 }  
 if (!newName.empty()) {  
 it->name = move(newName);  
 }  
 } else {  
 newName = encode(newName);  
 if (ranges::any\_of(programs, [&](const show& s) {  
 return s.name == newName && s.name != name; })) {  
 cout << "Show with this name already exists." << endl;  
 return;  
 }  
 it->name = move(newName);  
 }  
  
 if (newCategory.empty()) {  
 cout << "Category: ";  
 getline(cin, newCategory);  
 newCategory = encode(newCategory);  
 if (!newCategory.empty()) {  
 it->category = move(newCategory);  
 }  
 } else {  
 newCategory = encode(newCategory);  
 it->category = move(newCategory);  
 }  
  
 if (newStartTime.empty()) {  
 cout << "Start Time (HH:MM): ";  
 getline(cin, newStartTime);  
 }  
  
 if (!newStartTime.empty()) {  
 try {  
 // Find the position of the colon  
 size\_t colonPos = newStartTime.find(':');  
 if (colonPos != string::npos) {  
 // Extract hours and minutes  
 string hourStr = newStartTime.substr(0, colonPos);  
 string minStr = newStartTime.substr(colonPos + 1);  
  
 // Convert to integers  
 int hour = stoi(hourStr);  
 int minute = stoi(minStr);  
  
 // Validate the time  
 if (hour >= 0 && hour <= 23 && minute >= 0 && minute <= 59) {  
 it->startHour = hour;  
 it->startMinute = minute;  
 } else {  
 cout << "Invalid time values. Hours must be 0-23, minutes 0-59." << endl;  
 }  
 } else {  
 cout << "Invalid time format. Use HH:MM format." << endl;  
 }  
 } catch (const exception& e) {  
 cout << "Error parsing time: " << e.what() << ". Using original time." << endl;  
 }  
 }  
  
 if (!newDuration) {  
 cout << "Duration: ";  
 string durationStr;  
 getline(cin, durationStr);  
  
 if (!durationStr.empty()) {  
 try {  
 newDuration = stoi(durationStr);  
 if (newDuration <= 0) {  
 cout << "Invalid duration. Please provide a positive value." << endl;  
 } else {  
 it->duration = newDuration;  
 }  
 } catch (const exception& e) {  
 cout << "Error parsing duration: " << e.what() << endl;  
 }  
 }  
 } else {  
 it->duration = newDuration;  
 }  
  
 if (newDayOfWeek.empty()) {  
 cout << "Day of Week: ";  
 getline(cin, newDayOfWeek);  
 if (!newDayOfWeek.empty()) {  
 it->dayOfWeek = move(newDayOfWeek);  
 }  
 } else {  
 it->dayOfWeek = move(newDayOfWeek);  
 }  
  
 if (newChannelCode.empty()) {  
 cout << "Channel Code: ";  
 getline(cin, newChannelCode);  
 if (!newChannelCode.empty()) {  
 // Check if channel code exists  
 bool channelExists = ranges::any\_of(channels,  
 [&newChannelCode](const channel& c) { return c.code == newChannelCode; });  
 if (!channelExists) {  
 cout << "Error: Channel code does not exist. Channel not updated." << endl;  
 return;  
 }  
 it->channelCode = move(newChannelCode);  
 }  
 } else {  
 // Check if channel code exists  
 bool channelExists = ranges::any\_of(channels,  
 [&newChannelCode](const channel& c) { return c.code == newChannelCode; });  
 if (!channelExists) {  
 cout << "Error: Channel code does not exist. Channel not updated." << endl;  
 return;  
 }  
 it->channelCode = move(newChannelCode);  
 }  
  
 if (!fileExists("Program.txt")) {  
 cout << "File not found. Cannot update." << endl;  
 return;  
 }  
 ofstream o("Program.txt");  
 for (const auto& s : programs) {  
 string startTimeStr = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 o << s.name << " " << s.category << " " << startTimeStr << " " << s.duration << " " << s.dayOfWeek << " " << s.channelCode << endl;  
 }  
 o.close();  
  
 cout << "Show updated successfully." << endl;  
 } else {  
 cout << "Show not found." << endl;  
 }  
}  
  
void editChannel(const string& name, string newName, string newOriginCountry) {  
 if (name.empty()) {  
 cout << "Invalid input. Please provide valid channel details." << endl;  
 return;  
 }  
  
 string encName = encode(name);  
 auto it = ranges::find\_if(channels, [&encName](const channel& c) { return c.name == encName; });  
 if (it != channels.end()) {  
 cout << "Editing channel: " << decode(it->name) << endl;  
  
 if (newName.empty() || newOriginCountry.empty()) {  
 cout << "Enter new details (leave blank to keep current value):" << endl;  
 }  
  
 if (newName.empty()) {  
 cout << "Name: ";  
 getline(cin, newName);  
 newName = encode(newName); // Fix: encode newName, not name  
 if (ranges::any\_of(channels.begin(), channels.end(), [&](const channel& c) {  
 return c.name == newName && c.name != it->name; })) {  
 cout << "Channel with this name already exists." << endl;  
 return;  
 }  
 if (!newName.empty()) {  
 it->name = move(newName);  
 }  
 } else {  
 newName = encode(newName);  
 if (ranges::any\_of(channels, [&newName, &it](const channel& c) {   
 return c.name == newName && c.name != it->name; })) {  
 cout << "Channel with this name already exists." << endl;  
 return;  
 }  
 it->name = move(newName);  
 }  
  
 if (newOriginCountry.empty()) {  
 cout << "Origin Country: ";  
 getline(cin, newOriginCountry);  
 if (!newOriginCountry.empty()) {  
 newOriginCountry = encode(newOriginCountry);  
 it->originCountry = move(newOriginCountry);  
 }  
 } else {  
 newOriginCountry = encode(newOriginCountry);  
 it->originCountry = move(newOriginCountry);  
 }  
  
 if (!fileExists("Channel.txt")) {  
 cout << "File not found. Cannot update." << endl;  
 return;  
 }  
 ofstream o("Channel.txt");  
 for (const auto& c : channels) {  
 o << c.code << " " << c.name << " " << c.originCountry << endl;  
 }  
 o.close();  
  
 cout << "Channel updated successfully." << endl;  
 } else {  
 cout << "Channel not found." << endl;  
 }  
}  
  
void broadcastSummary() {  
 if (channels.empty() || programs.empty()) {  
 cout << "No channels or shows available." << endl;  
 return;  
 }  
  
 if (!fileExists("BroadcastSummary.txt")) {  
 createFileIfNotExists("BroadcastSummary.txt");  
 }  
 ofstream o("BroadcastSummary.txt");  
  
 // Map to store channel names and show counts  
 map<string, int> channelCounts;  
  
 // Count shows for each channel  
 for (const auto& show : programs) {  
 // Find the channel name for this show  
 for (const auto& channel : channels) {  
 if (channel.code == show.channelCode) {  
 channelCounts[channel.name]++;  
 break;  
 }  
 }  
 }  
  
 // Write results to file  
 for (const auto& [channelName, count] : channelCounts) {  
 o << channelName << " " << count << endl;  
 }  
  
 o.close();  
 cout << "Broadcast summary has been written to BroadcastSummary.txt" << endl;  
}  
  
void specificDayShow(const string& day) {  
 vector<show> sortedShows;  
  
 // Convert input day to lowercase  
 string dayLower = day;  
 ranges::transform(dayLower, dayLower.begin(), ::tolower);  
  
 // Case-insensitive day matching  
 for (auto &s : programs) {  
 string programDayLower = s.dayOfWeek;  
 ranges::transform(programDayLower, programDayLower.begin(), ::tolower);  
  
 if (programDayLower == dayLower) {  
 sortedShows.push\_back(s);  
 }  
 }  
  
 if (sortedShows.empty()) {  
 cout << "No shows found for the specified day." << endl;  
 return;  
 }  
  
 // Sort shows by start time  
 ranges::sort(sortedShows, [](const show& a, const show& b) {  
 if (a.startHour != b.startHour) {  
 return a.startHour < b.startHour;  
 }  
 return a.startMinute < b.startMinute;  
 });  
  
 // First pass: determine needed column widths based on content  
 int nameWidth = 4; // minimum width for "Name"  
 int categoryWidth = 8; // minimum width for "Category"  
 int timeWidth = 10; // minimum width for "Start Time"  
 int durationWidth = 8; // minimum width for "Duration"  
 int channelWidth = 12; // minimum width for "Channel Code"  
  
 // Determine maximum content width for each column  
 for (const auto& s : sortedShows) {  
 nameWidth = max(nameWidth, static\_cast<int>(decode(s.name).length()));  
 categoryWidth = max(categoryWidth, static\_cast<int>(decode(s.category).length()));  
 channelWidth = max(channelWidth, static\_cast<int>(s.channelCode.length()));  
  
 // Calculate duration string length and consider it for column width  
 string durationStr = to\_string(s.duration) + " min";  
 durationWidth = max(durationWidth, static\_cast<int>(durationStr.length()));  
  
 // Calculate time string length  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 timeWidth = max(timeWidth, static\_cast<int>(startTime.length()));  
 }  
  
 // Add padding (1 space on each side)  
 nameWidth += 2;  
 categoryWidth += 2;  
 timeWidth += 2;  
 durationWidth += 2;  
 channelWidth += 2;  
  
 // Print header  
 int totalWidth = nameWidth + categoryWidth + timeWidth + durationWidth + channelWidth + 6; // 6 for the separators  
 cout << endl << "Shows on " << day << ":" << endl;  
 cout << string(totalWidth, '-') << endl;  
 cout << "|" << setw(nameWidth) << left << " Name"  
 << "|" << setw(categoryWidth) << " Category"  
 << "|" << setw(timeWidth) << " Start Time"  
 << "|" << setw(durationWidth) << " Duration"  
 << "|" << setw(channelWidth) << " Channel Code" << "|" << endl;  
 cout << string(totalWidth, '-') << endl;  
  
 // Print data rows  
 for (const auto& s : sortedShows) {  
 string name = decode(s.name);  
 string category = decode(s.category);  
 string channelCode = s.channelCode;  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 string durationStr = to\_string(s.duration) + " min";  
  
 cout << "|" << setw(nameWidth) << " " + name  
 << "|" << setw(categoryWidth) << " " + category  
 << "|" << setw(timeWidth) << " " + startTime  
 << "|" << setw(durationWidth) << " " + durationStr  
 << "|" << setw(channelWidth) << " " + channelCode << "|" << endl;  
 }  
  
 cout << string(totalWidth, '-') << endl;  
 cout << sortedShows.size() << " shows found." << endl;  
}  
  
void maxShow() {  
 if (programs.empty()) {  
 cout << "No shows available." << endl;  
 return;  
 }  
  
 int maxDuration = 0;  
 vector<show> longestShows;  
  
 // First, find the maximum duration  
 for (const auto& s : programs) {  
 if (s.duration > maxDuration) {  
 maxDuration = s.duration;  
 }  
 }  
  
 // Then collect all shows with that duration  
 for (const auto& s : programs) {  
 if (s.duration == maxDuration) {  
 longestShows.push\_back(s);  
 }  
 }  
  
 // First pass: determine needed column widths based on content  
 int nameWidth = 4; // minimum width for "Name"  
 int categoryWidth = 8; // minimum width for "Category"  
 int timeWidth = 10; // minimum width for "Start Time"  
 int durationWidth = 8; // minimum width for "Duration"  
 int dayWidth = 3; // minimum width for "Day"  
 int channelWidth = 12; // minimum width for "Channel Code"  
  
 // Determine maximum content width for each column  
 for (const auto& s : longestShows) {  
 nameWidth = max(nameWidth, static\_cast<int>(decode(s.name).length()));  
 categoryWidth = max(categoryWidth, static\_cast<int>(decode(s.category).length()));  
 dayWidth = max(dayWidth, static\_cast<int>(decode(s.dayOfWeek).length()));  
 channelWidth = max(channelWidth, static\_cast<int>(s.channelCode.length()));  
  
 // Calculate duration string length and consider it for column width  
 string durationStr = to\_string(s.duration) + " min";  
 durationWidth = max(durationWidth, static\_cast<int>(durationStr.length()));  
  
 // Calculate time string length  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 timeWidth = max(timeWidth, static\_cast<int>(startTime.length()));  
 }  
  
 // Add padding (1 space on each side)  
 nameWidth += 2;  
 categoryWidth += 2;  
 timeWidth += 2;  
 durationWidth += 2;  
 dayWidth += 2;  
 channelWidth += 2;  
  
 // Print header  
 int totalWidth = nameWidth + categoryWidth + timeWidth + durationWidth + dayWidth + channelWidth + 7; // 7 for the separators  
 cout << endl << "Shows with the longest duration (" << maxDuration << " minutes):" << endl;  
 cout << string(totalWidth, '-') << endl;  
 cout << "|" << setw(nameWidth) << left << " Name"  
 << "|" << setw(categoryWidth) << " Category"  
 << "|" << setw(timeWidth) << " Start Time"  
 << "|" << setw(durationWidth) << " Duration"  
 << "|" << setw(dayWidth) << " Day"  
 << "|" << setw(channelWidth) << " Channel Code" << "|" << endl;  
 cout << string(totalWidth, '-') << endl;  
  
 // Print data rows  
 for (const auto& s : longestShows) {  
 string name = decode(s.name);  
 string category = decode(s.category);  
 string day = decode(s.dayOfWeek);  
 string channelCode = s.channelCode;  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 string durationStr = to\_string(s.duration) + " min";  
  
 cout << "|" << setw(nameWidth) << " " + name  
 << "|" << setw(categoryWidth) << " " + category  
 << "|" << setw(timeWidth) << " " + startTime  
 << "|" << setw(durationWidth) << " " + durationStr  
 << "|" << setw(dayWidth) << " " + day  
 << "|" << setw(channelWidth) << " " + channelCode << "|" << endl;  
 }  
  
 cout << string(totalWidth, '-') << endl;  
 cout << longestShows.size() << " shows found." << endl;  
}  
  
void minShow() {  
 if (programs.empty()) {  
 cout << "No shows available." << endl;  
 return;  
 }  
  
 int minDuration = INT\_MAX;  
 vector<show> shortestShows;  
  
 // First, find the minimum duration  
 for (const auto& s : programs) {  
 if (s.duration < minDuration) {  
 minDuration = s.duration;  
 }  
 }  
  
 // Then collect all shows with that duration  
 for (const auto& s : programs) {  
 if (s.duration == minDuration) {  
 shortestShows.push\_back(s);  
 }  
 }  
  
 // First pass: determine needed column widths based on content  
 int nameWidth = 4; // minimum width for "Name"  
 int categoryWidth = 8; // minimum width for "Category"  
 int timeWidth = 10; // minimum width for "Start Time"  
 int durationWidth = 8; // minimum width for "Duration"  
 int dayWidth = 3; // minimum width for "Day"  
 int channelWidth = 12; // minimum width for "Channel Code"  
  
 // Determine maximum content width for each column  
 for (const auto& s : shortestShows) {  
 nameWidth = max(nameWidth, static\_cast<int>(decode(s.name).length()));  
 categoryWidth = max(categoryWidth, static\_cast<int>(decode(s.category).length()));  
 dayWidth = max(dayWidth, static\_cast<int>(decode(s.dayOfWeek).length()));  
 channelWidth = max(channelWidth, static\_cast<int>(s.channelCode.length()));  
  
 // Calculate duration string length and consider it for column width  
 string durationStr = to\_string(s.duration) + " min";  
 durationWidth = max(durationWidth, static\_cast<int>(durationStr.length()));  
  
 // Calculate time string length  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 timeWidth = max(timeWidth, static\_cast<int>(startTime.length()));  
 }  
  
 // Add padding (1 space on each side)  
 nameWidth += 2;  
 categoryWidth += 2;  
 timeWidth += 2;  
 durationWidth += 2;  
 dayWidth += 2;  
 channelWidth += 2;  
  
 // Print header  
 int totalWidth = nameWidth + categoryWidth + timeWidth + durationWidth + dayWidth + channelWidth + 7; // 7 for the separators  
 cout << endl << "Shows with the shortest duration (" << minDuration << " minutes):" << endl;  
 cout << string(totalWidth, '-') << endl;  
 cout << "|" << setw(nameWidth) << left << " Name"  
 << "|" << setw(categoryWidth) << " Category"  
 << "|" << setw(timeWidth) << " Start Time"  
 << "|" << setw(durationWidth) << " Duration"  
 << "|" << setw(dayWidth) << " Day"  
 << "|" << setw(channelWidth) << " Channel Code" << "|" << endl;  
 cout << string(totalWidth, '-') << endl;  
  
 // Print data rows  
 for (const auto& s : shortestShows) {  
 string name = decode(s.name);  
 string category = decode(s.category);  
 string day = decode(s.dayOfWeek);  
 string channelCode = s.channelCode;  
 string startTime = (s.startHour < 10 ? "0" + to\_string(s.startHour) : to\_string(s.startHour)) + ":" +  
 (s.startMinute < 10 ? "0" + to\_string(s.startMinute) : to\_string(s.startMinute));  
 string durationStr = to\_string(s.duration) + " min";  
  
 cout << "|" << setw(nameWidth) << " " + name  
 << "|" << setw(categoryWidth) << " " + category  
 << "|" << setw(timeWidth) << " " + startTime  
 << "|" << setw(durationWidth) << " " + durationStr  
 << "|" << setw(dayWidth) << " " + day  
 << "|" << setw(channelWidth) << " " + channelCode << "|" << endl;  
 }  
  
 cout << string(totalWidth, '-') << endl;  
 cout << shortestShows.size() << " shows found." << endl;  
}  
  
void averageShow(const string& category) {  
 int sum = 0, count = 0;  
 for (auto& s : programs) {  
 if (s.category == encode(category)) {  
 sum += s.duration;  
 count++;  
 }  
 }  
 if (count == 0) {  
 cout << "No shows available in the " << category << " category." << endl;  
 } else {  
 double average = static\_cast<double>(sum) / count;  
 cout << "Average duration of shows in category " << category << ": " << average << " minutes." << endl;  
 }  
}  
  
void showMenu() {  
 int choice = 0;  
 string name, category, dayOfWeek, channelCode, originCountry;  
 int duration;  
  
 do {  
 cout << "\n===== TV Program Management System =====" << endl;  
 cout << "1. Show all shows" << endl;  
 cout << "2. Show all channels" << endl;  
 cout << "3. Add show" << endl;  
 cout << "4. Add channel" << endl;  
 cout << "5. Delete show" << endl;  
 cout << "6. Delete channel" << endl;  
 cout << "7. Edit show" << endl;  
 cout << "8. Edit channel" << endl;  
 cout << "9. Make a Broadcast Summary" << endl;  
 cout << "10. Show shows on a specific day" << endl;  
 cout << "11. Show longest show" << endl;  
 cout << "12. Show shortest show" << endl;  
 cout << "13. Average show" << endl;  
 cout << "14. Exit" << endl;  
 cout << "Enter your choice: ";  
  
 string input;  
 getline(cin, input);  
  
 // Try to convert the input to an integer  
 try {  
 choice = stoi(input);  
 } catch (const exception&) {  
 cout << "Invalid input. Please enter a number." << endl;  
 cout << "\nPress Enter to continue...";  
 cin.get();  
 clearScreen();  
 continue; // Skip the rest of the loop iteration  
 }  
  
 switch (choice) {  
 case 1:  
 clearScreen();  
 allShows();  
 break;  
 case 2:  
 clearScreen();  
 allChannels();  
 break;  
 case 3: {  
 clearScreen();  
 string startTime;  
 cout << "Enter show name: ";  
 getline(cin, name);  
 cout << "Enter category: ";  
 getline(cin, category);  
 cout << "Enter start time (HH:MM): ";  
 getline(cin, startTime);  
 cout << "Enter duration (minutes): ";  
 getline(cin, input);  
 try {  
 duration = stoi(input);  
 } catch (const exception&) {  
 cout << "Invalid duration. Operation cancelled." << endl;  
 cout << "\nPress Enter to continue...";  
 cin.get();  
 clearScreen();  
 continue;  
 }  
 cout << "Enter day of week: ";  
 getline(cin, dayOfWeek);  
 cout << "Enter channel code: ";  
 getline(cin, channelCode);  
 addShow(name, category, startTime, duration, dayOfWeek, move(channelCode));  
 break;  
 }  
 case 4:  
 clearScreen();  
 cout << "Enter channel name: ";  
 getline(cin, name);  
 cout << "Enter origin country: ";  
 getline(cin, originCountry);  
 addChannel(name, originCountry);  
 break;  
 case 5:  
 clearScreen();  
 cout << "Enter name of show to delete: ";  
 getline(cin, name);  
 deleteShow(name);  
 break;  
 case 6:  
 clearScreen();  
 cout << "Enter name of channel to delete: ";  
 getline(cin, name);  
 deleteChannel(name);  
 break;  
 case 7:  
 clearScreen();  
 cout << "Enter name of show to edit: ";  
 getline(cin, name);  
 editShow(name);  
 break;  
 case 8:  
 clearScreen();  
 cout << "Enter name of channel to edit: ";  
 getline(cin, name);  
 editChannel(name);  
 break;  
 case 9:  
 clearScreen();  
 broadcastSummary();  
 break;  
 case 10:  
 clearScreen();  
 cout << "Enter day of week: ";  
 getline(cin, dayOfWeek);  
 specificDayShow(dayOfWeek);  
 break;  
 case 11:  
 clearScreen();  
 maxShow();  
 break;  
 case 12:  
 clearScreen();  
 minShow();  
 break;  
 case 13:  
 clearScreen();  
 cout << "Enter category name: ";  
 getline(cin, category);  
 averageShow(category);  
 break;  
 case 14:  
 clearScreen();  
 cout << "Exiting program. Goodbye!" << endl;  
 break;  
 default:  
 cout << "Invalid choice. Please try again." << endl;  
 break;  
 }  
   
 if (choice != 14) {  
 cout << "\nPress Enter to continue...";  
 cin.get();  
 clearScreen();  
 }  
 } while (choice != 14);  
}

# Datele de intrare ( Emisiuni.txt; Canal.txt )

Program.txt:

Stirile\_de\_seara Stiri 19:00 60 Luni 1

Vocea\_tinerei\_generatii Talent\_show 20:30 120 Marti 1

Film\_de\_actiune Film 22:30 110 Miercuri 2

Telenovela\_latina Telenovela 13:00 45 Joi 1

Campionatul\_national Sport 21:00 120 Vineri 3

Gustos\_si\_rapid Cooking\_show 20:00 90 Luni 4

Povesti\_adevarate Documentar 18:00 60 Duminica 1

Satul\_vesel Comedie 20:30 30 Joi 1

Actualitati Stiri 19:00 60 Sambata 4

Talent\_romanesc Talent\_show 20:00 180 Sambata 4

Dialoguri\_deschise Talk\_show 17:00 60 Luni 1

Bucataria\_lui\_chef Cooking\_show 16:00 30 Miercuri 5

Planeta\_albastra Documentar 21:00 60 Joi 6

Provocarea\_finala Reality\_show 20:00 100 Marti 2

Grand\_prix Sport 15:00 90 Duminica 3

Aventuri\_magice Film 20:00 150 Sambata 7

Cinema\_de\_seara Film 22:00 100 Marti 8

Istorie\_ascunsa Documentar 19:30 60 Luni 6

Campionii\_dansului Talent\_show 20:00 120 Vineri 4

Ora\_de\_comedie Comedie 21:30 30 Miercuri 2

Serial\_politist Serial 22:00 60 Marti 21

Desene\_animate Animatie 08:00 120 Sambata 24

Istoria\_neamului Documentar 19:00 60 Duminica 9

Fitness\_acasa Lifestyle 07:00 30 Luni 27

Show\_culinar Cooking\_show 14:00 60 Marti 5

Interviuri\_cu\_vedete Talk\_show 16:30 45 Miercuri 27

Calatorie\_in\_jurul\_lumii Documentar 20:00 60 Joi 20

Fashion\_week Lifestyle 19:00 30 Vineri 29

Natura\_salbatica Documentar 21:00 60 Sambata 18

Filme\_horror Film 23:00 120 Duminica 8

Channel.txt:

1 ProTV Romania

2 Kanal\_D Romania

3 DigiSport1 Romania

4 Antena1 Romania

5 Paprika Ungaria

6 National\_Geographic Statele\_Unite

7 HBO Statele\_Unite

8 Cinemax Statele\_Unite

9 TVR1 Romania

10 TVR2 Romania

11 Prima\_TV Romania

12 Antena3 Romania

13 DigiSport2 Romania

14 DigiSport3 Romania

15 Eurosport Franta

16 Eurosport2 Franta

17 Discovery Statele\_Unite

18 Animal\_Planet Statele\_Unite

19 History\_Channel Statele\_Unite

20 Travel\_Channel Statele\_Unite

21 AXN Statele\_Unite

22 FilmCafe Ungaria

23 Disney\_Channel Statele\_Unite

24 Cartoon\_Network Statele\_Unite

25 BBC\_Earth Marea\_Britanie

26 CNN Statele\_Unite

27 ZU\_TV Romania

28 MTV Statele\_Unite

29 Fashion\_TV Franta

30 Comedy\_Central Statele\_Unite

# Date de ieșire (BroadcastSummary.txt)

BroadcastSummary.txt:

AXN 1

Animal\_Planet 1

Antena1 4

Cartoon\_Network 1

Cinemax 2

DigiSport1 2

Fashion\_TV 1

HBO 1

Kanal\_D 3

National\_Geographic 2

Paprika 2

ProTV 6

TVR1 1

Travel\_Channel 1

ZU\_TV 2

# Capturi de ecran cu reprezentare lucrului programului

A screen shot of a program

AI-generated content may be incorrect.

1. Meniul programului

În captura de ecran putem vedea toate funcțiile programei care le putem accesa prin intermediului numerelor de la 0 la 14 pur și simplu introducând cifra. Primele două funcții sunt destul de simple de aceea nu au nevoie de lămurire.

Funcția cu numărul 3 și 4 sunt destul de simple și ele pur simplu citesc de la tastatura informația care o introducem după care înscrie în fișierele corespunzătoare emisiunile și respectiv canalele noi. Informațiile noi trebuie să corespundă condițiilor practicii.

Capturi:

A screen shot of a computer program

AI-generated content may be incorrect.

1. Adăugarea unei emisiuni noi

A black screen with white text

AI-generated content may be incorrect.

1. Adăugarea unui nou canal

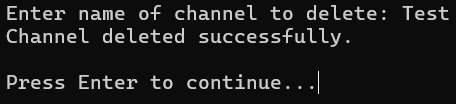
Funcția 5 și 6 sunt două funcții care ne ajută să ștergem din fișierele cu emisiuni sau canale informația despre un element.

Capturi:

A black background with white text

AI-generated content may be incorrect.

1. Ștergerea unei emisiuni



1. Ștergerea unui canal

Funcțiile 7 și 8 sunt folosite pentru a schimba informațiile unei emisiuni/canal existent cu altele noi. Schimbarea propagându-se la fișierele lor respective.

Capturi :

A screen shot of a computer program

AI-generated content may be incorrect.

1. Editarea unei emisiuni existente

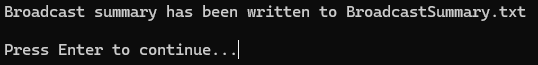
A screenshot of a computer program

AI-generated content may be incorrect.

1. Editarea unui canal existent

Funcția 9 este folosită pentru a crea fișierul BroadcastSummary.txt care va conține fiecare canal și numărul de emisiuni difuzate de acel canal.

Captură:



1. Crearea fișierului BroadcastSummary.txt

Funcția 10 caută după ziua săptămânii introdusă de utilizator toate emisiunile din acea zi.

Captura:

A screenshot of a computer

AI-generated content may be incorrect.

1. Afișarea emisiunilor după ziua introdusă de utilizator

Funcțiile 11 și 12 prin intermediul comparației caută emisiunea cu durata minimă și respectiv cea maximă din Program.txt.

Capturi:

A screen shot of a computer

AI-generated content may be incorrect.

1. Afișarea celor mai lungi emisiuni

A screenshot of a computer

AI-generated content may be incorrect.

1. Afișarea celor mai scurte emisiuni

Funcția 13 preia de la utilizator o categorie care o folosește mai apoi pentru a calcula durata medie a tuturor emisiunilor din acea categorie.

Captura:

A black screen with white text

AI-generated content may be incorrect.

1. Afișarea duratei medie a tuturor emisiunilor dintr-o categorie specificată de utilizator

Și în final funcția 14 – funcția care va închide programul.

# Concluzie

În urma parcurgerii etapelor propuse în cadrul stagiului de practică, s-au obținut rezultate semnificative atât din punct de vedere tehnic, cât și al dezvoltării profesionale. Am reușit:

* **Sinteza rezultatelor**:
  + Implementarea unui modul C++ pentru gestionarea emisiunilor și canalelor TV, care respectă toate cerințele de citire, scriere și actualizare în fișiere text, a demonstrat aplicabilitatea învățăturilor teoretice în contexte reale.
  + În Excel, importul datelor, configurarea listelor derulante, adăugarea comentariilor și generarea grafice a statisticilor zilnice au adus o înțelegere practică a instrumentelor de analiză și vizualizare a datelor.
  + În Word și PowerPoint, structurarea raportului și a prezentării a consolidat abilitățile de redactare profesională și comunicare vizuală.
* **Complexitatea și limitările cercetării**:
  + Proiectul a implicat manipularea concurențială a fișierelor text și gestionarea excepțiilor (date lipsă, formate incorecte), ceea ce a ridicat probleme de consistență și securitate a datelor.
  + În Excel, sincronizarea automată între tabele și actualizarea dinamică a sumelor pentru canale s-a dovedit sensibilă la modificările manuale, iar generarea graficelor a necesitat ajustări repetitive de format.
* **Direcții pentru cercetări viitoare**:
  + **Dezvoltarea unei aplicații cu interfață grafică (GUI)**, care să permită gestionarea emisiunilor și canalelor fără acces direct la fișiere.
  + **Migrarea datelor într-un SGBD (de exemplu, SQLite sau MySQL)**, pentru a asigura integritate tranzacțională și interogări mai flexibile.
* **Comentariu personal**:  
  Obiectivele enunțate în Introducere—evaluarea competențelor de student în primul an și aplicarea practică a cunoștințelor din programarea și testarea produselor program—au fost îndeplinite în mare măsură. Am dobândit o viziune clară asupra provocărilor reale: gestionarea coerentă a datelor, planificarea sarcinilor, colaborarea inter funcțională și respectarea calendarului de livrare. În același timp, am constatat că perfecționarea continuă (în special în domeniul bazelor de date și al interfețelor utilizator) este esențială pentru a transforma o soluție de laborator într-un produs robust și scalabil.

# Bibliografie

Dascal, A. (2022). *Programare procedurală în C++.* Preluat de pe Elibrary CEITI: https://elibrary.ceiti.md/book/programare-procedurala-in-c

Dascal, A. (2022). *Programare structurată în C++.* Preluat de pe Elibrary CEITI: https://elibrary.ceiti.md/book/programare-structurata-in-c

Dascal, A. (2022). *Programare structurată și procedurală în C++.* Preluat de pe Elibrary CEITI: https://elibrary.ceiti.md/book/programare-structurata-si-procedurala-in-c