# Shopping Management System

| TEAM MEMBERS | | |
|---|---|---|
| **SL NO** | **NAME** | **REGD NO** |
| 1. | Ashirwad Shekhar | CL2025010601919829 |
| 2. | Apoorb Raj | CL20250106019422126 |
| 3. | Ashish Kumar | CL20250106018922104 |
| 4. | Aryabrat Sahoo | CL2025010601919728 |

## Introduction

The Shopping Management System is a console-based application written in C++ that simulates a simple online shopping experience. The system leverages a CSV file named Products.csv—which stores the name and price of each product—to dynamically load available items. Users can add products to a shopping cart, view a detailed bill, and remove items as needed. This project served as a practical learning exercise to deepen understanding of key C++ concepts and to explore the integration of file handling with object-oriented design.

## Problems with Existing Shopping Management Systems

Before developing our solution, several issues were identified with traditional shopping management systems:

- Limited Data Management: Many existing systems lacked efficient mechanisms to manage and update product data dynamically. Often, product information was hardcoded or manually updated, leading to inconsistencies.

- Poor User Interaction: Conventional systems sometimes suffered from a non-intuitive user interface, making it cumbersome for users to add or remove products from the shopping cart.

- Inadequate Error Handling: Early versions of shopping systems rarely provided robust error handling for invalid user inputs, resulting in system crashes or incorrect billing.

- Scalability Challenges: Without proper data structures and file management techniques, expanding the product catalog and maintaining real-time updates proved to be a challenge.

## Solution with Our Shopping Management System

Our system addresses these challenges by:

- Dynamic Data Loading: The product information is stored in a CSV file (Products.csv), allowing for easy updates and scalability. The system reads the file at startup, converting each line into a Product object.

- Modular Object-Oriented Design: By employing classes such as Product and Bill, the system encapsulates data and functionality, promoting code reusability and modular design.

- Efficient Data Structures: A C++ vector is used to manage the shopping cart, providing dynamic storage that can grow as more products are added.

- Robust Input and Error Handling: The application includes mechanisms to handle invalid inputs gracefully. For example, it validates user choices in the menu and utilizes file handling checks to ensure that the CSV file is properly opened before reading.

- Interactive Menu System: A simple yet effective menu-driven interface guides users through actions like adding products, displaying the bill, and removing products from the cart.

These features collectively offer a more robust, user-friendly, and scalable solution compared to traditional approaches.

## Libraries Used and Their Descriptions

The following C++ libraries are integral to the functionality of the system:

- iostream: Used for standard input and output operations, enabling communication with the user through the console.

- string: Provides support for string manipulation, which is essential for handling product names and file input.

- limits: Facilitates handling of numeric limits, especially useful for clearing input errors and validating user data.

- vector: Offers dynamic array functionality, used here to manage the list of products in the shopping cart efficiently.

- fstream: Enables file input/output operations, crucial for reading the product data from the Products.csv file.

- sstream: Allows for string stream operations, which are used to parse individual lines of the CSV file into product attributes.

Each library plays a critical role in ensuring the system operates smoothly while reinforcing core C++ programming concepts.

# Code:

```cpp
#include <iostream>
#include <string>
#include <limits>
#include <vector>
#include <fstream>
#include <sstream>

using namespace std;

//Product class
class Product {
private:
    string name;
    double price;

public:
    Product(string n, double p): name(n), price(p) {}
    string getName() const {return name;}
    double getPrice() const {return price;}

};

class Bill {
private:
    vector<Product> cart;

public:
    void addProduct(const Product& product) {
        cart.push_back(product);
    }

    void displayBill() {
        if (cart.empty()) {
            cout << "Bill Empty. Please add items to the bill." << endl;
            return;
        }

        double amt = 0;
        cout << "Items:" << endl;
        for (const Product& product: cart) {
            cout << "Name: " << product.getName() << ", Price: Rs." << product.getPrice() <<
endl;
            amt += product.getPrice();
        }
        cout << "Total Amount: Rs." << amt << endl << endl;
    }

    void removeProduct(string product) {
        if (cart.empty()) {
            cout << "Bill Empty. No products can be removed from it." << endl;
            return;
        }
        vector <Product>::iterator delIndex;
        for (delIndex = cart.begin(); delIndex != cart.end(); ++delIndex) {
            if (delIndex->getName() == product) {
                cart.erase(delIndex);
                cout << "Product removed from bill." << endl;
                return;
            }
        }
        cout << "Product does not exist in the bill." << endl;
    }
};

bool isProductExists(const string& productName, const vector<Product>& productList) {
    for (const Product& product: productList) {
        if(product.getName() == productName) {
            return true;
        }
    }
    return false;
}

vector<Product> readCSV(const string& filename) {
    vector<Product> productList;
    ifstream file(filename);

    if (file.is_open()) {
        string line;
        while(getline(file, line)) {
            stringstream ss(line);
            string name;
            double price;
            if(getline(ss, name, ',') && ss >> price) {
                Product product(name, price);
                productList.push_back(product);
            }
        }
        file.close();
    } else {
        cerr << "Error Opening File: " << filename << endl;
    }
    return productList;
}

void displayMenu() {
    cout << "Main Menu:" << endl;
    cout << "1. Add Product to Cart" << endl;
    cout << "2. Show Bill" << endl;
    cout << "3. Remove Product from Cart" << endl;
    cout << "0. Exit" << endl;
}

void menuProduct(Bill& bill, vector<Product>& productList) {
    string name;
    double price;
    cout << "Enter the Product name: ";
    cin >> name;
    if (isProductExists(name, productList)) {
        for (const Product& product: productList) {
            if(product.getName() == name) {
                bill.addProduct(product);
                cout << name << " added successfully to the bill!" << endl;
                return;
            }
        }
    } else {
        cout << "Product not found. Please try again." << endl;
    }
}

void removeProduct(Bill& bill) {
    string delProduct;
    cout << "Enter Product to remove: ";
    cin >> delProduct;
    bill.removeProduct(delProduct);
}

int main() {
    string productFile = "Products.csv";
    vector<Product> productList = readCSV(productFile);

    int choice;
    static Bill bill;

    while(true) {
        menu:
        displayMenu();
        cout<< "Enter your Choice: ";
        if (cin >> choice) {
            switch (choice) {
                case 1:
                    menuProduct(bill, productList);
                    break;
                case 2:
                    bill.displayBill();
                    break;
                case 3:
                    removeProduct(bill);
                    break;
                case 0:
                    cout << "Exiting....." << endl;
                    cout << "Thank you for using our services!"<< endl;
                    exit(0);
                default:
                    cout << "Invalid Input! Please Try Again." << endl;
            }
        } else { //Error Handling of input choice
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Invalid Datatype Error!!!!" << endl;
            goto menu;
        }
    }
}
```

## Future Scope

While the current system successfully demonstrates core functionalities, several enhancements can be explored in the future:

- Graphical User Interface (GUI): Transitioning from a console-based application to a GUI would significantly improve user interaction and accessibility.
- Enhanced Exception Handling: Incorporating more sophisticated try-catch mechanisms and custom exceptions could further improve robustness, especially in handling file I/O errors.
- Extended Product Management: Future versions might include additional product attributes (such as description, category, and stock quantity) and support for dynamic updates to the product catalog.
- Database Integration: Integrating a database would enable better management of large inventories, real-time updates, and multi-user support.
- Security Features: Implementing user authentication and secure transaction handling would make the system more viable for real-world e-commerce applications.
- Online Payment Integration: Including modules for online payment processing could transform the system into a full-fledged e-commerce solution.

## Conclusion

The development of the Shopping Management System in C++ provided invaluable experiential learning by integrating various programming paradigms and libraries. By addressing the limitations of traditional shopping management systems—such as static data handling and limited error management—the project demonstrated how modern C++ techniques can be applied to build a more efficient, user-friendly solution. The project not only reinforced the use of vectors, classes, file handling, and exception handling but also paved the way for future enhancements that could transform it into a comprehensive e-commerce system.

Overall, this project has been a critical step in understanding and applying C++ programming concepts in a real-world context, setting a strong foundation for further learning and development in software engineering.